NTNU – Trondheim
Norwegian University of
Science and Technology

Title

Vegard Stjerna Lindrup

Submission date: June 2016
Responsible professor: Tor Engebret Onshus
Supervisor:

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Title: Title
Student: Vegard Stjerna Lindrup

Problem description:

Dette legges til i DAIM, og blir derfor fjernet før innlevering.

Responsible professor: Tor Engebret Onshus
Supervisor:

Abstract

Mobile robot platforms etc...

Sammendrag

Mobile robotplatformer kan kjøre rundt og...

Preface

Hva synes jeg om oppgaven? Kjempeartig! Eget acknowledgemet-kapittel?

Acknowledgments

This thesis, and the obtained project results would not have been possible without help from student colleagues, friends and support from the Department of Engineering Cybernetics.

I would first like to thank my project supervisor, Professor Tor Onshus of the Department of Engineering Cybernetics at NTNU, for allowing me to work on such an open and interesting topic, and for providing valuable advice and guidance through the two last semesters. He has been quick to respond to problems with the project, and gave me a much needed sense of urgency when the project was lagging far behind schedule in march.

Among my student colleagues, Eirik Wold Solnør, Vegard Blomseth Johnsen and Henrik Rudi Haave have been particularly helpful during testing sessions and for video documentation. Over the last two semesters, Ole Magnus Siqveland and I have used the same robot platform for our projects. Through good collaboration, we found a hardware setup that worked for both of us; a new shelf structure with compartments for the various hardware components.

I would like to thank the foreman, Terje Haugen, and apprentice Daniel Bogen at the mechanical workshop for building the new compartments and frames for the robot used in this thesis. Many thanks goes to the employees at the electronics workshop for allowing me to borrow tools and equipment, and providing some hints and tips.

I am very grateful to my parents and Andrea Myklebust for supporting me through the years at NTNU. Thank you!

Sincerely,
Vegard Stjerna Lindrup

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

AIV Autonomous Inspection Vehicle

CLM Concurrent Localization and Mapping

CP Cathodic Protection

DRC DARPA Robotics Challenge

EKF Extended Kalman Filter

Fraunhofer IPA Fraunhofer Institute for Manufacturing Engineering and Automation

GUI Graphical User Interface

HMI Human-Machine Interaction

HSE Health, Safety and Environment

IFR International Federation of Robotics

IO Integrated Operations

ISS International Space Station

ITK Department of Engineering Cybernetics

LIDAR LIght Detection And Ranging

MAR Mobile Autonomous Robot

MIMROex Mobile Inspection and Monitoring Robot, experimental

MIT Massachusetts Institute of Technology

NCS Norwegian Continental Shelf

NDT Non-destructive Testing

NUI Natural user interface

OCS Operator Control Station

O&G Oil & Gas

OpenCV Open Source Computer Vision Library

ORB Oriented FAST and Rotated BRIEF

PCL Point Cloud Library

PR Personal Robot

PWM Pulse Width Modulation

RBI Risk Based Inspection

ROS Robot Operating System

ROV Remotely Operated Vehicle

RPAS Remotely Piloted Aerial System

RTAB-Map Real-Time Appearance-Based Mapping

SDF Simulation Description Format

SIFT Scale-invariant feature transform

SIM Structural Integrity Management

SLAM Simultaneous Localization And Mapping

SSD Solid State Drive

SURF Speeded Up Robust Features

STAIR Stanford AI Robot

TLA Three Letter Acronym

UAV Unmanned Aerial Vehicle

URDF Unified Robot Description Format

VR Virtual Reality

List of Algorithms

Chapter 1
Introduction

1.1 About the Project

This thesis presents and documents this authors master's project, **TTK4900**, on robotic maintenance which was carried out at the Department of Engineering Cybernetics (ITK) in the spring of 2016. **TTK4900** is worth 30 credits (studiepoeng), and the project duration is set to 22 full-time weeks with the possibility of extension in case of a valid reason. The work presented in this thesis is carried out as independent work which is supervised by Professor Tor Onshus through regular status meetings.

1.1.1 The Topic - Mobile Autonomous Robot

The robot system that was used in this project has been developed over the course of many preceding master and specialization projects. The long term goal of these projects is to develop mobile autonomous robot concepts for maintenance and inspection on topside offshore installations. The topic is given by Professor Tor Onshus at Department of Engineering Cybernetics (ITK). A description of this topic[1], suggests some possible applications for such a robot:

- The robot could serve in a supporting role as a part of Integrated Operations (IO).
- It can also be used to prepare a normally unmanned topside offshore installation before the arrival of a maintenance crew, by performing safety checks and preparing the helicopter landing pad.
- Allow personnel to perform remote inspection and maintenance through telepresence.
- In combination with Virtual Reality (VR), the robot could be used for training purposes.

[1] http://folk.ntnu.no/onshus/Oppgaver.htm

1.2 Preceding Projects

Telepresence and Robotic Arm

The system in its current form is built around a robot manipulator arm, SCORBOT-ER4u. Kristian Saxrud Bekken focused on improving previous work on the system, which was done as early as 2005[Bek10]. Bekken's work comprise telepresence through a stereo video transmission, a collision avoidance system for the robot arm and an improved Human-Machine Interaction (HMI) implementation.

Building the Mobile Platform

During the spring of 2013, Petter Aspunvik devoted his master's project to develop a mobile base for the robotic arm[Asp13]. Aspunvik's thesis has served as a user manual for many of the robot systems in the early stages of this project. The current motor control firmware used Aspunvik's implementation as a starting points.

Simultaneous Localization and Mapping

In parallel to Aspunvik's project, Mikael Berg developed a solution for Simultaneous Localization And Mapping (SLAM) and autonomous navigation for the same robot[Ber13]. His software is programmed in Google's Go language, and runs on Windows 7 within the pre-installed on-board computer. The resulting system successfully utilized Hector SLAM with a LIDAR and odometry from two encoder wheels for 2D navigation and SLAM. Berg considered to create a solution based on ROS which requires a Linux platform. In the end, he opted to target the Windows platform as this is the only operating system which is compatible with the robotic arm. The "future work"-section in Berg's thesis suggests improvements in the form of 3D obstruction detection, because the LIDAR is limited to detection in a plane. He also mentions object recognition and dynamic re-planning as possible extensions.

Last Year's Specialization Project

This author's specialization project[Lin15] presented an obstruction detector based on two unsynchronized IP-cameras and a stereo matching algorithm in Open Source Computer Vision Library (OpenCV). Because the cameras were unsynchronized, the system would become useless whenever there is relative motion between the robot and the surroundings. The obstruction detector lacked a critical feature: a floor filter to separate the ground from potential obstructions. The implementation presented in this master's project is unrelated to the preceding specialization project, except for some useful c++ functions.

1.3 Implementation Overview

1.3.1 Selecting Tools and Hardware

To meet objective 4 in the problem description, it was decided to focus on vision based navigation. A robot with the ability to build a map of the surroundings and relocate autonomously was considered to be a good starting point for further vision based solutions. As a continuation of this author's specialization project, the robot was equipped with a 3D camera, a Kinect for XBOX 360 capable of perceiving depth images at a high frame rate ($30Hz$).

To use the Kinect, the initial plan was to utilize the Point Cloud Library (PCL) in combination with a SLAM method (e.g. Kintinous[Kind]). This approach came with a high degree of uncertainty that would reduce the project scope significantly, and increase the likelihood of an unsatisfactory result. The Robot Operating System (ROS), an open source robot software framework, came up as an alternative tool late in January.

The work and solutions presented in this thesis revolves around the process of integrating ROS with the mobile robot from [Asp13] and [Ber13]. Installing ROS on Ubuntu Linux is by far the easiest way to begin using the framework. For this reason, and to avoid interfering with another project on the same robot, it was decided that an additional computer running Linux should be fitted to the robot. The mobile robot from [Asp13] and [Ber13] was refitted to accommodate the Kinect and the second computer.

Two supporting tools were implemented in addition to the robot software: a simple concept of an OCS and a hand held remote control implemented on a smartphone. An overview of the complete system is shown in figure 1.1.

1.3.2 Functionality

1. Simultaneous localization and mapping based on computer vision.
2. Mapping over multiple sessions.
3. Autonomous navigation to a simple goal.
4. 3D and 2D obstacle avoidance in navigation mode.
5. The robot can be controlled by an Android Smartphone.
6. The robot can stream video to an URL.
7. The robot can receive velocity commands over WiFi.

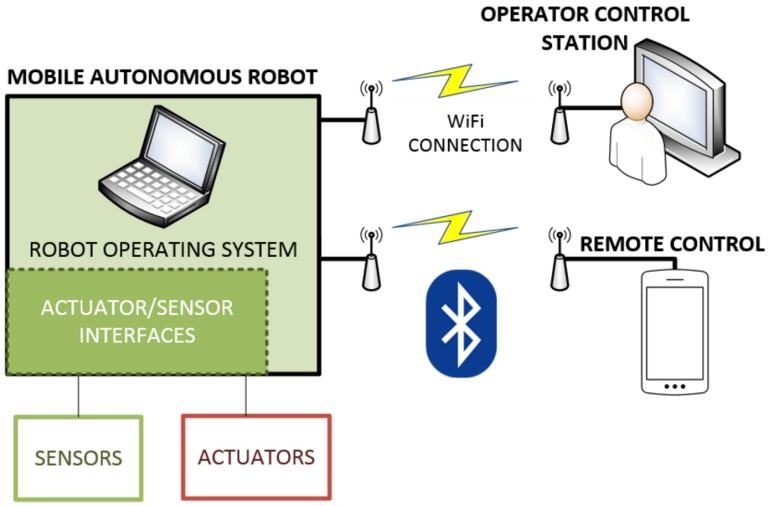


Figure 1.1: System Concept. An on-board computer using ROS to handle actuators and sensors. Remote operation is available through an OCS or a hand-held device with Bluetooth.

1.4 Thesis Structure

Chapter 2
Robotic Maintenance on Topside Offshore Platforms

2.1 Introduction

This project is a small step towards a larger long-term goal concerning robotic maintenance on topside offshore installations. This section puts the following background theory, and the implementation described in chapter 4, into the context of the long-term goal.

Over the last decade, the Oil & Gas (O&G) industry has shown an increased interest in the potential benefits of automating the normal operation of remote offshore installations. Some satellite platforms, such as Sleipner B are already unmanned during normal operation, and the more recent Valemon platform is planned to become normally unmanned as well. It is now clear that robotics has several potential applications in process plants, and particularly in remote O&G installations. It is, however, difficult to predict how and to what extent robots will be applied in plant automation, as other innovative solutions may arise[sta][sub][E24]. Current research on plant automation is mainly motivated by two factors[KMP15] [AS12]:

- **HSE** - Reduced risk exposure for personnel and environment.
- **Efficiency** - Accomplish more with less effort, resources and time. This means cost reduction by keeping downtime to a minimum with the least amount of effort.

An additional overarching driving factor is that O&G fields are becoming more difficult to reach. As O&G fields in shallow waters are depleted, production is moved to deeper waters. This complicates the extraction process and reduces the profit margin. A solution to this challenge is to increase efficiency through further automation. To what extent automation will be in the form of robots, or other innovative solutions is difficult to say at this time.

Robot Applications Including Categorization		
Category	Robot	Robot Task/Activity Description
B	Pipeline rigging robot	To autonomously load and offload pigs into pipelines.
C	Boat handling robot < 500 kg	To transfer personnel and loads below 500 kg to and from boats.
D	Boat handling robot > 500 kg	To transfer loads above 500 kg to and from boats.
B	Mobile universal service robot version 1	To perform "buddy" roles; carrying, holding, lifting, personal safety monitoring etc.
C	Mobile universal service robot version 2	To autonomously perform task not involving manipulation of the process or facilities.
D	Mobile universal service robot version 3	To autonomously perform tasks involving manipulation of the process or facilities.
D	Treatment/Inspection robot	To autonomously perform inspection/treatment(painting) tasks of structures/vessels or facilities.
A	Domestic service robot	To autonomously perform floor cleaning, catering, laundry handling, storage handling and logistics activities.

Table 2.1: Some tasks with potential to be "robotized", and the corresponding robot category[PBB11].

This section provides a brief introduction to how topside maintenance is performed today, how these maintenance tasks could be robotized. The section is concluded with a discussion on how well modern robotics is suited for the task.

2.2 Robotizing Offhsore Maintenance

A feasibility study performed by researchers from Fraunhofer Institute for Manufacturing Engineering and Automation (Fraunhofer IPA)[PBB11] identified several topside production tasks and ranked them according to their resource demand. Based on the identified tasks, the study went on to describe a set of specific tasks, and then assess how easy or hard it would be to "robotize" these tasks. Table 3.1 associates a set of tasks with different robot categories, as well as how easy or hard the process of "robotizing" the activity is expected to be. The difficulty levels are described

with the letters ”A” through ”D”, where ”A” is described as of the shelf robotics, which makes ”robotizing” easy. Activities associated with the letter ”D” on the other hand, cannot be be ”robotized” with either current or near future technology even if doing so would be beneficial [PBB11]. Note that the paper in question is from 2011, and the difficulty of these tasks may have changed. This is particularly true in the domain of visual sensors, given the bloom of accessible 3D sensor technology and research over the last decade. Some further elaboration on inspection activities and environmental considerations is given in the next subsection.

2.2.1 Structural Maintenance and Environmental Considerations

Corrosion

Offshore installations are regularly, if not continuously, exposed to harsh weather conditions in the form of wind and seawater. Presence of seawater, either through direct contact or in the form of drops and vapor, forms a very corrosive environment. The offshore and marine environment is classified as the most corrosive environment in ISO 12944[ER12a]. It is essential to provide countermeasures to ensure safe and reliable operation over the lifetime of the installation. Common corrosion prevention methods are[ER12a]:

- Sacrificial Anodes.
- Cathodic Protection (CP) in the form of a DC-current.
- Protective coating.

In terms of maintenance, the sacrificial anodes can be subjected to periodic inspections and replacements, which could be done by a robot. CP can more easily be implemented with automated self tests, and should normally not require any inspections and maintenance[ER12a]. Application of protective coating should ideally be applied in the controlled environment of a workshop. If protective coating is to be applied at sea, one should strive to make the conditions as favourable as possible.

Structural Fatigue

Waves, wind, water currents and other forces subject offshore installations to structural stress. To keep the offshore installations from failing in these conditions, they may be subjected to a Risk Based Inspection (RBI) regime. In brief, RBI is a strategy where inspection and maintenance programs are developed based on which risk factors an installation is exposed to. In an automated maintenance program, an autonomous robot could perform inspections of the structure and generate reports based on risk factors such as[ER12b]:

Marine growth at sea level Marine growth will increase the diameter of supporting legs at sea level, thus increasing structural loads caused by waves, wind and water currents.

Corrosion Assess the seriousness of a corrosion attack through Non-destructive Testing (NDT).

Scour Scour around the platform legs could reduce a platforms ability to withstand structural loads. This is only applicable to non-floating installations.

A maintenance expert can then plan a maintenance campaign based on data from the robot in combination with knowledge on the platforms design, age and exposure to the environment.

2.2.2 Production-specific Hazards

O&G production has several inherent hazards, and an unwanted incident may have serious implications for Health, Safety and Environment (HSE) and production uptime. The Piper Alpha incident serves as a worst case example of the consequences of an explosive ignition of a hydrocarbon leak followed by an escalating fire. This section will briefly discuss some of the most significant hazards on an offshore oil and gas production plant.

Hydrocarbon leaks

Hydrocarbon leaks do occur on a regular basis. Over a four year period from 2006 to 2010, seven leaks larger than $1kg/s$ of either oil or gas/two-phase occurred in the Norwegian sector. No such leaks have ignited on the Norwegian Continental Shelf (NCS) since 1992. Of all the leaks which occurred in the same area, NCS, the majority was caused by human intervention[Vin14]. This could imply that a reliable robotic system may reduce the number of leaks.

Fire

Critical fire loads[1] on offshore facilities are usually caused by uncontrolled flow of hydrocarbons. The most serious of such releases is a blow-out. Risk reducing measures focus on four areas[Vin14]:

Leak prevention - Use equipment and assembly methods which minimize risk.

[1]Fire load can be defined as the amount of combustible material in a given area (Ref. https://en.wiktionary.org/wiki/fire_load).

Leak detection - Fire & gas detection, emergency shut-down systems and blow-down systems.

Ignition prevention - Inspection and maintenance and Ex-approved equipment.

Escalation protection - Installation layout and sectioning. Fire and gas protection systems.

Explosions

Explosion protection is usually built into the equipment and structure. In [Vin14], there are no obvious ways a robot could provide additional explosion beyond e.g. leak detection, inspection and maintenance.

2.2.3 Implications for Robot Design

A mobile robot operating on a normally unmanned platform in a harsh environment, implies that it is subjected to many of the same design philosophies that apply to subsea equipment.

Because of the risk of explosive atmospheres in an offshore production environment, an offshore robot operating under EU or EEA legislation will also be subject to the ATEX (ATmosphères EXplosibles) directive. Such a robot will most likely carry ignition sources such as batteries packed with energy and perhaps even welding equipment. A central ATEX requirement is to perform a risk assessment. As outlined by ATEX 2014-34-EU Guidelines[ATE], such a risk assessment is usually performed in four steps:

1. **Hazard identification** What can go wrong? Identify possible ignition sources, and the probability of explosive atmospheres.

2. **Risk estimation** Estimate the probability of an unwanted occurrence (e.g. an explosion), and the associated consequences.

3. **Risk evaluation** Evaluate the identified risk in context of acceptable risk, and decide if the design should be altered or if additional barriers should be installed. Barriers could either mitigate the consequences of an explosion, or reduce the possibility of ever having an explosion.

4. **Risk reduction option analysis** Identify possible risk reduction measures, e.g. barriers and design changes. A cost-benefit analysis can be performed in accordance with the ALARP-principle.

The on-board embedded computer hardware and software should be designed for robustness, fault tolerance and endurance. If a failure occurs, corrective actions will most likely be both difficult and expensive. Resistance to corrosion and toxic environments should also be taken into account.

Other design choices, e.g. the shape, size and equipment must fit the robots purpose. A source of guidelines for a mobile robot can draw upon case studies such as e.g. [GP08].

2.3 Robotic Maintenance Today

2.3.1 Trends and Potential

The typical pre-programmed assembly robots still dominate the robotic market. They are usually found in manufacturing plants and large scale production facilities[ifr], e.g. the automotive industry, where they perform dull, tedious tasks much faster and with higher accuracy than people. A notable trend in modern robotics is increased human-robot collaboration[Bog16]. Many new robots are being build for the human workspace, both in terms of safety and collaborative functionality. This trend is a step along the way of moving robots out of the controlled environment of a factory floor, and into the real world where a high degree of autonomy is required.

A report by Metra Martech[GC11], a market research firm referenced to by International Federation of Robotics (IFR)[2], points to three areas with a high potential for robotic applications:

- Dangerous jobs, e.g. handling dangerous materials or work in high risk environments.
- Jobs that are economically infeasible in a high wage economy.
- Work which is impossible or highly inconvenient for humans, e.g. space exploration, subsea maintenance or assembly of heavy components.

All of these factors motivate the development of robots for autonomous robotic maintenance.

2.3.2 Subsea Maintenance and Inspection

[2] http://www.ifr.org/robots-create-jobs/

Figure 2.1: Subsea 7's AIV. This is the first commercial autonomous inspection vehicle for subsea operations [pre]

Subsea maintenance is perhaps the field that have seen the greatest advancements in autonomous inspection and maintenance. As offshore installations are moved to the seabed, maintenance and inspection has become a significant challenge. This has resulted in a widespread use of Remotely Operated Vehicles (ROVs). Recent developments in other fields, e.g. computer vision, human-robot collaboration and machine learning, has resulted in new Autonomous Inspection Vehicles (AIVs) and Autonomous Underwater Vehicles (AUVs) capable of performing inspection and simple maintenance tasks autonomously[JWA+12][RCR+15]. A driving factor behind the transition from ROVs to AUVs is cost reduction through increased offshore campaign efficiency.

2.3.3 Disaster Response

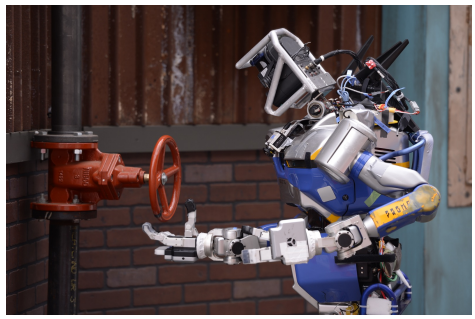


Figure 2.2: Team HRP2-Tokyo's robot turning a valve during DARPA Robotics Challenge 2015 (Image credits: DARPA Robotics Challenge)

Robots in disaster response, relief and recovery solve many of the same problems faced by maintenance robots. Disasters, such as the tsunami which struck Japan in 2011, proved that much work needs to be done, both in terms of technical capabilities and logistical issues related to deployment and response times. The tsunami resulted in three core meltdowns at the Fukushima Daiichi Nuclear Power plant.

Many of the robots which were deployed at the Fukushima Power Plant were already ageing, and the operators had to receive training before deployment, thus increasing the response time[KFO12]. A paper from Japan Atomic Energy Agency[KFO12] highlights how the lack of stakeholder involvement could have been the cause of long response times. The same paper points out that the robots were developed for the sake of development,

and not with emergency response as the main purpose[KFO12].

DARPA Robotics Challenge (DRC)[DRC] was launched in response to the Fukushima disaster of 2011. The purpose of the competition is to accelerate innovation, research and development in robotics for disaster response in cases where humans cannot operate. Some of the tasks the competitors faces in 2015 include valve turning, traversing rubble and driving a vehicle through a course before egressing out of the vehicle.

2.3.4 Topside Offshore and Onshore Robotic Maintenance

Figure 2.3: An early version of the maintenance robot "Sensabot", developed by National Robotic Engineering Center (NREC) (Image credits: NREC)

Today, autonomous and teleoperated inspection and maintenance is usually only found at subsea installations. Topside installations on the other hand are still maintained and inspected manually, with some notable exceptions. Small Unmanned Aerial Vehicles (UAVs) or Remotely Piloted Aerial Systems (RPAS) have become commonplace and accessible to all over the last decade. There are currently RPAS systems which are being used for visual inspection of inaccessible structural parts such as flare stacks or the exterior of oil rigs.

Some notable contributors to the field of robotic maintenance for O&G include ABB, Fraunhofer IPA, Sintef ICT[KLT09] and NREC at Carnegie Mellon University.

NRECs contribution, Sensabot, is a remotely operated inspection robot designed for harsh and remote environments[dep12]. It is not designed to be autonomous, but rather as a tool to move personnel from hazardous environments to safe remote control rooms. Sensabot mark II will be certified for zone 1 explosive environments. This year (2016), the plan is to test the robot on site at the Kashagan field in Kazakhstan[PSM+16].

Fraunhofer IPA[3] has developed a robot, called Mobile Inspection and Monitoring Robot, experimental (MIMROex). MIMROex has capabilities which are quite similar to the prototype used during the work on this thesis. MIMROex is equipped with a

[3] http://www.ipa.fraunhofer.de/en.html

camera for visual inspections as well as microphones, vibration and sensors for fire and gas detection. It is also certifiable in accordance with the explosion protection standard IEC 60079[MIM]. Fraunhofer IPA put great emphasis on field testing on actual offshore installations.

Both ABB and Sintef ICT has developed lab facilities to test various concepts for robotic maintenance. Both facilities use non-mobile robots which utilize a rich set of inspection and manipulation tools, as well as HMI equipment for remote operation and control. SINTEF has d The two research communities differ in that ABB has tested their solutions in real environments, which subjects their solution to ATEX requirements and an extensive risk management regime[AS12].

Another effort towards robotic maintenance is the ARGOS challenge (Autonomous Robot for Gas and Oil Sites). The purpose of the challenge is to promote innovation, understanding and awareness towards robotic maintenance of O&G sites in harsh environments[ARG].

Chapter 3
Background Theory

3.1 Modelling and Simulation

3.1.1 Some Terminology

Coordinate Systems and Poses

Robot Joints

All links are connected to each other by joints.

Coordinate systems are essential in the field of robotics.

3.1.2 Robot Modelling

3.1.3 Simulating in Gazebo

3.2 ROS

3.2.1 Introduction

The ROS is a collection of software libraries, tools and drivers intended for robot software development. A ROS installation can be tailored to meet the demands of a wide range of robots with varying complexity. ROS is usually installed in the form of an already built Debian-package. These packages are only compatible with a few versions of Ubuntu which are specified on the ROS homepage. When installed and configured, ROS will run on top of Linux, and can be perceived as and extention of Linux itself. Installing ROS from source is possible, but not recommended [ROSe].

Historic roots of ROS can be traced back to Stanford University at the beginning of the 2000s. At Stanford, several robotics software frameworks, including Stanford AI Robot (STAIR) and the Personal Robot (PR) program, were created to provide dynamic, flexible and well tested foundations for further robot development and research. In

2007, a nearby start-up company and robot incubator, Willow Garage, sought to build upon these concepts, and initiated a collaborative and open development process of a new software framework. This framework eventually became ROS[ROSb][QGS15]. The framework can be used under the BSD open-source license[**?**]. Today, ROS comes in many forms and comprise hundreds of advanced packages, algorithms and drivers, making it applicable for hobbyists, industrial automation, research and everything in between.

3.2.2 Important ROS Concepts

The following descriptions are included in order to provide a complete, self-contained description of the project implementation. Similar descriptions can be found on the official ROS website[1], as well as in any book on ROS (for example [QGS15]).

The ROS Graph

A ROS system comprise a set of small programs that communicate with each other through messages. These programs become nodes in the ROS graph. The nodes communicate with each other by publishing and subscribing to topics that form the edges of the graph. A topic must have the format of one of the specific data types provided by ROS. For example, a node which receives temperature data from a thermometer, may publish the data as a topic on the ROS system with the type `sensor_msgs/Temperature`. There are many other data formats, e.g. velocity messages, `geometry_msgs/Twist`; images, `sensor_msgs/Image`; odometry messages, `nav_msgs/Odometry` and so on. Each node in the graph are typically POSIX processes, and the edges are TCP connections[QGS15]. A minimal example of a graph is shown in figure 3.4.

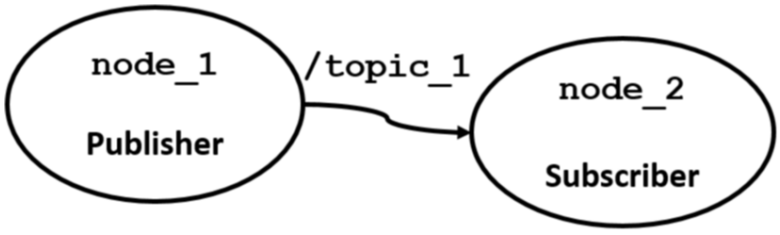


Figure 3.1: A minimal ROS graph. There are two nodes, `node_1` and `node_2`. `node_1` publishes data, i.e. a topic, by the name `topic_1`. `node_2` can receive the data by subscribing to `topic_1`.

[1] http://www.ros.org/

roscore

`roscore` is an essensial part of any ROS system as it enables nodes to communicate with each other. An instance of `roscore` must be started before launching any nodes. When a node is started, it will inform `roscore` of which topics it publishes and which topics it wish to subscribe to. Then, `roscore` will provide the information which allows the node to form a peer-to-peer connection to other nodes.

tf

`tf`[Foo13] is a coordinate system transformation library used in ROS. Parts of a ROS system can listen to broadcasted transforms

Project Structure and the *catkin* Build System

The source code in a ROS system is organized into packages. Each package provides a specific functionality to the system. Some packages can be downloaded and installed from a remote repository, while other packages will be created by the in-house developers for their specific robotic system. In this project, locally created ROS-packages were placed into a *catkin workspace*. This workspace contains the original source code and build specifications. Details are provided in chapter 4. As presented in[ROSa], a general workspace structure is as follows:

```
workspace_folder/        -- CATKIN WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

A ROS project will usually utilize the catkin build system.

roslaunch

`roslaunch`[ROSh] is a ROS package tool used to launch multiple nodes from a single command line. This is useful for larger projects with many nodes, interactions and parameters. Exactly which nodes to launch is defined in XML-files with the *.launch* extension. In a launch file, the developer can group nodes together, pass arguments

to the nodes and launch other launch files. Launch files can be launched from the command line as follows:

```
$ roslaunch <package name> <launch file name>.launch <argument1>:=true
```

3.2.3 An Overview of ROS-Related Tools

Robot Modelling In URDF

Unified Robot Description Format (URDF) is an XML-like format for describing robots. The robot description is made up of links and joints. Each link description contains information of, e.g., its shape, inertial tensor, collision boundaries. The links are connected to each other by joints.

Visialization in `rviz`

`rviz` is an invaluable tool for visualizing on-line robot behavior. Simply put, `rviz` is created to visualize what the robot sees, and how it plans ahead. Many of the images in the following chapters were obtained in `rviz`.

Simulation in Gazebo

3.2.4 Notable Robots Running ROS

PR2 - Personal Robot 2 PR2, developed by Willow Garage is one of the first robots designed to run ROS [QGS15], and also one of the most advanced and capable robots with ROS today. PR2 is build for research and development of service robot applications. The navigation stack used in this thesis has been tested on the PR2. [MEBF+10] describes how the PR2 used the navigation stack to autonomously navigate 42 km (26.2 miles). PR2 is available for sale at the price of $280,000.00[2] (2016).

TurtleBot TurtleBot is a cheaper ROS-ready alternative to PR2. It is consists of a mobile base with differential drive, and a shelf system for mounting laptop computers and sensors.

Robonaut 2 Robonaut 2[3], a dexterous humanoid robot, currently resides within the International Space Station (ISS) 400 km above the earth's surface. In 2014, a SpaceX Dracon capsule brought ROS as well as a pair of legs for Robonaut up to the ISS[ROSg]. Robonaut is designed for research on how robots can support the crew in maintaining and operating the space station. A potential application of Robonaut

[2] https://www.willowgarage.com/pages/pr2/order
[3] ROS in space from ROSCon 2014: https://vimeo.com/106993914

is to perform extra vehicular activities and other maintenance tasks, thus freeing up valuable time for the crew.

Being the first robot with ROS to be launched into space,

Industrial Hardware The ROS-industrial program[ROSc] provides hardware interfaces to various industrial equipment. An example is ABB's IRB-2400, where ROS-industrial provides package for motion planning software (MoveIt!) and trajectory downloading[ROSd].

3.3 Software

3.3.1 Qt

3.3.2 PCL

3.4 The Kinect Sensor

3.5 Software Tools

3.5.1 Point Cloud Library

3.5.2 ROS

3.5.3 Qt

3.5.4 Current Research and Applications

3.6 Introduction to Sensors in Autonomous Robots

3.6.1 Depth Cameras

Different Methods for Depth Perception

A depth camera can be described as a regular color video camera with the ability to create spatial images. In the context of this thesis, a depth camera can more precisely be described as a RGB-D camera, where the letters RGB-D are short for red, green, blue and depth. In a regular RGB camera, a spatial scene will be projected onto a rectangular pixel grid where each pixel contains intensity values for red, green and blue colors. These pixel values represents the detected scene. A major problem with RGB cameras is the significant loss of information. The information loss is mostly a consequence of 3D to 2D projection and digital quantization. RGB-D cameras have the means to reduce this information loss by mapping the pixel values to spatial coordinates. The atomic parts in 3d images are usually represented as points in a point cloud or cubic volumes, also known as voxels.

Different variations of depth cameras will usually fall into one of two categories: active or passive. Passive sensors perceive the surroundings as it is, without actively interfering with the environment as a part of the sensing process. A typical passive RGB-D sensor is the stereo camera. Stereo cameras use a stream of synchronized image pairs to perceive depth. The image pairs are displaced along the horizontal axis, and the depth information is extracted by searching for mutual information in the image pairs. How far the information is displaced from the left to the right image is directly related to how far away from the camera the information source is located.

Active sensors depend on some form of projection onto the surroundings. For depth cameras, the projection is usually in the form of laser or infra red light. In RGB-D cameras it is essential that the projected light is distinguishable from the visible spectrum. The Kinect sensor used in this project is an example of an active RGB-D sensor. A proper introduction to the Kinect will follow shortly.

3.6.2 Kinect for Xbox 360

Kinect for Xbox 360 is the RGB-D sensor used in this project. The device was initially intended as a Natural user interface (NUI) for gaming and office applications, and was the first consumer grade sensor to utilize structured light. Possible use cases were inspired by early NUI research at Massachusetts Institute of Technology (MIT) and, later on, the science fiction movie Minority Report, where Tom Cruice interacts with a computer by using hand gestures [WA12]. The Kinect sensor is equipped with a depth sensor, a regular color camera, a microphone array and a tilt motor(figure 3.5). The color camera in combination with the depth sensor forms what is usually referred to as a RGB-D sensor, i.e. a combined color and depth camera (figure 3.5). This feature, combined with its relatively low cost and high accessibility has contributed to make the Kinect very popular in research projects related to SLAM and robotics. In the three first years since it's release in 2010, over 3000 papers in well-known journals and proceedings were devoted to research on the Kinect sensor. Roughly 500 of these papers focused on SLAM or 3d reconstruction[BMNK13]. Some of the other papers focused on some of the weaknesses with the sensor, such as detection of glass surfaces and having several sensors in the same area.

Today, the the Kinect for Xbox 360 has been succeeded by the Kinect for Xbox One, and is now considered to be a legacy device. Those considering to use the legacy Kinect should be aware of that it is becoming increasingly difficult, if not already impossible, to get hold of a new Kinect for Xbox 360.

Natural User Interfaces - Origin of the Kinect

The idea behind a NUI is to make the HMI as seamless and natural as possible. A NUI allows the user to communicate without tools such as a keyboard or a mouse.

For decades, NUIs have only existed as ideas, science fiction or research projects. This has changed dramatically over the last ten years, and NUIs can now be considered to be ubiquitous. Today, the most common form of NUIs is the touch screen found in smart phones and tablets.

The Microsoft Kinect sensor was initially designed as a NUI for the Xbox 360 gaming console. The sensor allows users to use gestures and sounds to play console games. Later on, Microsoft has released SDKs, enabling developers to create NUI applications for for Windows.

Kinect Hardware Specifications

Sensor documentation provided online by Microsoft is incomplete and untidy. The reason for this may be that the first Kinect is considered to be obsolete. The following specifications are based on [WA12] and the "Kinect for Windows Programming Guide"[kinc]. There is in fact a distinction between the Kinect for Windows and Kinect for XBOX 360. Only Kinect for Windows can be used in commercial applications[WA12]. Another important issue is that there are compatibility issues between ROS and Kinect for Windows. It is unknown to this author if the compatibility issues have been fixed, but possible hacks have been suggested by the community[kina][kinb].

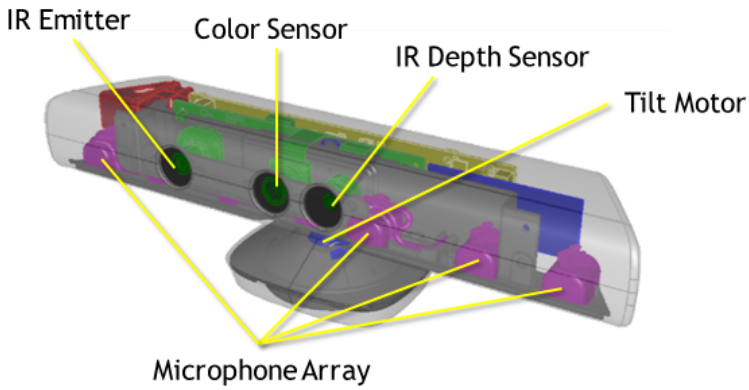


Figure 3.2: Kinect sensor components. (Image credits: Microsoft[kinc])

Sensor specifications are given in table 3.2. Note that the range values may differ from what is available in Microsoft's own SDK, as the shortest distance was measured manually in ROS. In [kinc], the depth values for Kinect for Xbox 360 range from $0.8m$ to $4m$. Other distances are either unknown, too close or too far away. Kinect for Windows can operate in "near mode", i.e. it can measure distances from $0.4m$ to $3m$.

Kinect for Xbox 360 Specifications	
Viewing Angle	43° vertical by 57° horizontal field of view.
Image Resolution	640x480
FPS	30 Hz (given 640x480 depth and color video)
Minimum Depth (measured)	$\approx 0.5m$.
Maximum Depth	$8m$.
Normal (Reliable) Depth Range	$0.8m < x < 4m$

Table 3.1: Kinect for Xbox 360 Specifications.

3.6.3 Planar Laser Sensors (LIDAR)

A planar laser sensor, known as e.g. laser proximity sensors or laser radars, can all be referred to as LIDARs.

Scanning Laser Range Finder, URG-04LX-UG01

3.6.4 Odometers

3.6.5 Sensor Fusion

3.7 Simultaneous Localization and Mapping (SLAM)

3.7.1 Introduction to SLAM

SLAM, also known as Concurrent Localization and Mapping (CLM), is a class of solutions to the problem of determining an agents location and pose in an unknown environment, while simultaneously mapping the same environment.

3.7.2 Hector SLAM

Hector SLAM [KMvSK11] is a 2D SLAM approach capable of 3D motion estimation. In its original form, the method is suitable for systems with low-end computational power and size, and for mapping of small environments. 2D SLAM is based on LIght Detection And Ranging (LIDAR) scans aligned with the horizontal plane. The full Hector SLAM implementation consist of of a 2D SLAM subsystem loosely coupled and synchronized with an Extended Kalman Filter (EKF) used as a 6DOF pose estimator.

The 2D pose of the LIDAR is estimated through a scan matcher, i.e. the process of aligning the current LIDAR scan with the map generated over the past time steps. The scan matcher was used to provide odometry to RTAB-Map, which is the chosen

SLAM system for this robot. Hector SLAM can be downloaded and used in ROS in the form of a prebuild package, `hector_slam`.

3.7.3 RTAB-Map

RTAB-Map is developed by IntRoLab at Université de Sherbrooke in Canada. It is a SLAM system developed for long term operations in large environments. The system is also intended to handle the "kidnapped robot-problem", i.e. multi-session mapping. This is useful whenever the robot is shut down and moved to an unmapped part of the same area, where it will start a new mapping session. RTAB-Map is the core feature that has been integrated into the robot described in this thesis. Some factors which motivated the use of RTAB-Map are:

- It is a SLAM method which requires an RGB-D sensor, for example a Kinect. The problem description for this project requires a vision based solution.
- RTAB-Map has a ROS wrapper, `rtabmap_ros`, which eases the process of integrating it with the mobile robot.
- It includes 3D obstacle detection.
- It has a memory management system intended for large scale multi-session mapping.
- RTAB-Map can be used for object detection. This can be done by linking RTAB-Map to OpenCV and the non-free feature detectors Scale-invariant feature transform (SIFT) and Speeded Up Robust Features (SURF).

The source code and ROS wrapper is currently maintained, and new features and bug-fixes are added regularly. RTAB-Map has two distinctive solutions to the SLAM problem: Visual loop closure detection and a memory management system for large data sets. The following paragraphs provides an overview of how RTAB-Map works. Detailed descriptions of the loop closure detection and memory management approach is provided in [LM13], while the SLAM method is presented in [LM14]. Further details can be found on the project's Github page[4].

Graph Based Mapping

RTAB-Map uses a graph structure with nodes and edges to represent the map. New nodes are continuously added to the system's working memory as time passes. In this method, the graph edges are referred to as *links*. There are two types of links:

[4] http://introlab.github.io/rtabmap/

neighbour links and loop closure links. Each node is a location in the map, and the links contain geometrical transformations between the node locations. Figure 3.6 illustrated the graph concept.

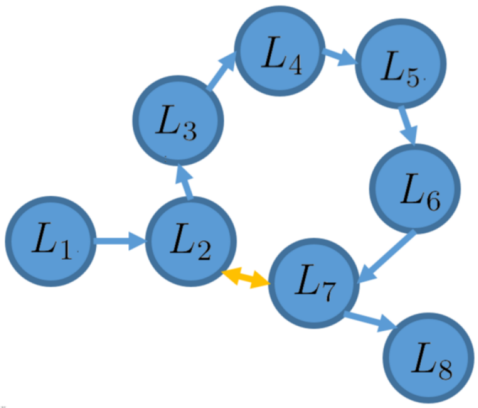


Figure 3.3: Conceptual illustration of a graph created by RTAB-Map over time $1 \leq t \leq 8$. A loop closure hypothesis was accepted at $t = 7$, as shown by the yellow arrow. Feature descriptors in L_2 and L_7 are sufficiently similar to accept this as a loop closure.

Loop Closure Detection

On-line Mapping of Large Environments

3.7.4 RGBD SLAM and Octomap

Octomap[HWB+13] is another 3D mapping framework available for ROS. Similar to RTAB-Map, Octomap can also be used as a standalone version.

Maps are represented by memory efficient Octrees where each leaf node represents a cube, or voxel, in the volumetric map. The voxel can be either occupied, free or unexplored. The volume of the cube is determined by how deep in the tree the leaf node is located. In a ROS graph, the Octomapping is performed by the node `octomap_server`. This node will subscribe to point cloud messages `sensor_msgs/PointCloud2`, and return volumetric occupancy maps, i.e. Octomaps.

There are several approaches to SLAM which uses Octomaps. An example that stands out in the context of ROS is[EHS+14]; a SLAM approach which depends on a RGB-D sensor, and relies on Octomap for efficient map storage.

This mapping framework was not used in this project in order to limit the project scope, and because the alternative RTAB-Map was associated with less uncertainty.

3.8 Autonomous Navigation

ROS has a built in navigation stack for 2D navigation. The navigation stack can plan a path and send velocity commands to the mobile base controller based on sensor input, a goal pose, a map and the frame of the `base_link`.

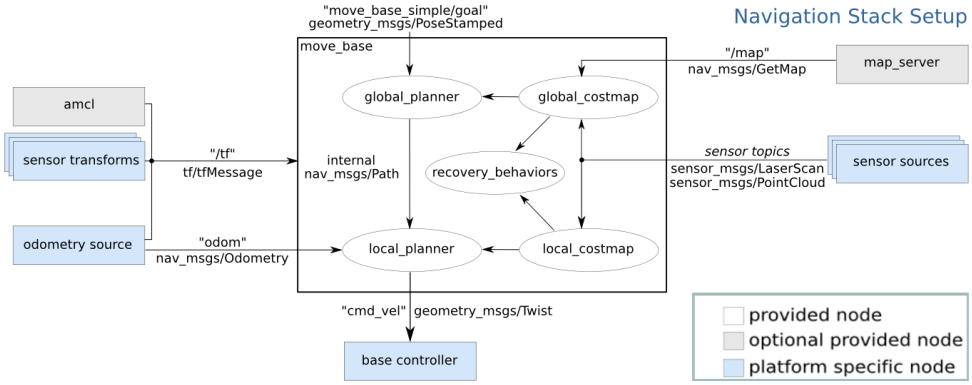


Figure 3.4: `move_base` and the navigation stack. (Image credits: ros.org)

Chapter 4
Implementation

4.1 Introduction

This chapter presents four implementations that were developed during this project:

Robot software in ROS	The robot's operating system runs on a laptop computer which is placed on the robot. This system is responsible for the core functionality of the robot, which includes sensor and actuator management, mapping, navigation and manual control.
Motor Control Firmware	Provides an interface between the ROS computer and each wheel motor. Translates velocity commands to wheel speeds.
Android Application	A supporting tool intended to function as a remote control for the robot. The implementation presented here enables the user to control the robot from an Android device via a Bluetooth connection.
Operator Control Station	A simple Operator Control Station (OCS) based on Qt enables an operator to control the robot via a wireless TCP/IP connection. The OCS can display a live video stream from the Kinect sensor.

4.2 Hardware Setup

The implementations above created a need for additional hardware. Extensive modifications had to be made to accommodate these additions. Some improvements with respect to safety were made as well.

4.2.1 Second On-Board Computer and New Rear Compartment

The robot was already fitted with an on-board computer running Windows 7. Using the on-board computer was not an option, due to the following reasons:

- It lacked the computational power required for this implementation. Both the Gazebo simulator and RTAB-Map is computationally intensive.
- The hard-drive was full. No disk space for the Linux partition required by ROS was available.
- Two parallel master's projects were using the robot hardware, and both project implementations required an on-board computer. Sharing the hardware would have been very time consuming.

For these reasons, this author decided to develop the robot software on a second on-board computer.

The previous chassis did not facilitate any good wiring solutions. The cables belonging to the various equipment on the robot would often result in a huge tangle of cables and wires. To accommodate the second on-board computer and to facilitate a tidier cable management, it was decided to build a new rear compartment where the equipment could be placed.

Figure 4.1: Sensor locations for LIDAR and Kinect.

4.2.2 Sensor Calibration and Setup

Only support for the Kinect and the LIDAR sensor was integrated into this system. Odometry from the two wheel encoders was not included because of time constraints

on the project. Figure 4.2 illustrates how the sensors and the wireless router is connected to the ROS computer and how they are supplied with power.

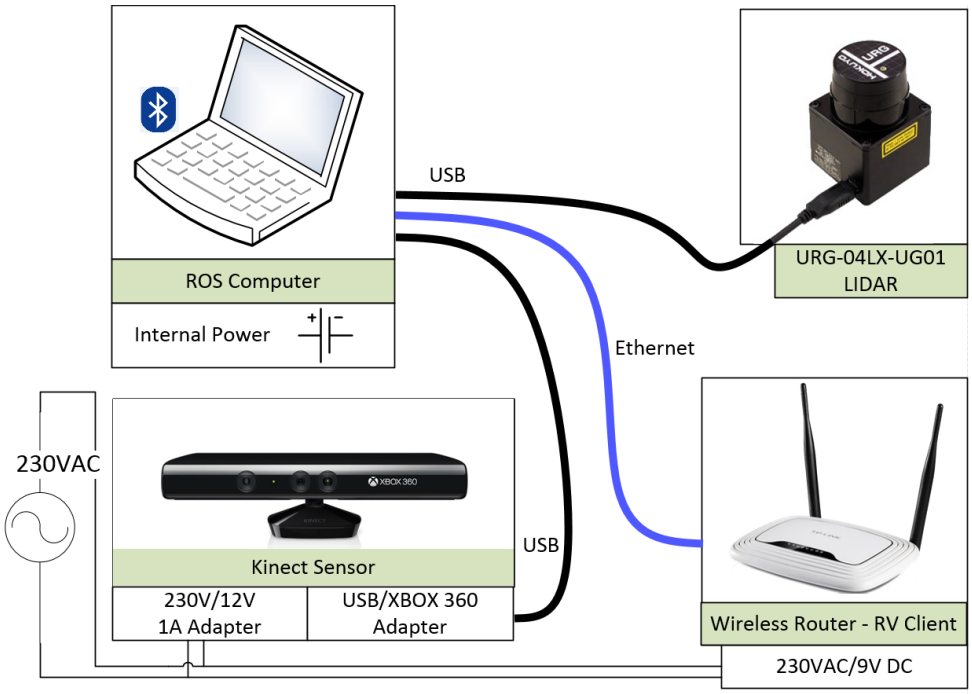
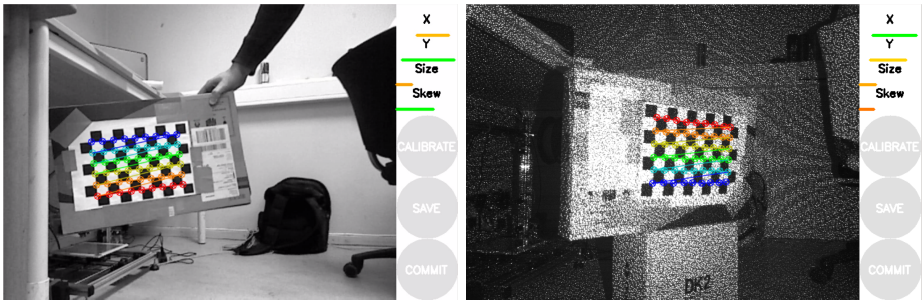


Figure 4.2: Sensor and power supply connections.

Calibrating both the Kinect and the LIDAR is a straight forward procedure with ROS. The Hokuyo LIDAR will in fact be calibrated automatically when the node is launched. Calibrating the Kinect is actually not strictly necessary because the lens distortion is very low. That being said, because there already is a calibration tool available in ROS[1] that is easy to use, there is no good reason to not calibrate. The calibration procedure is as follows:

1. Print out a chessboard pattern and tape it to a flat surface. It is beneficial to use A3 paper size to make the pattern easier to detect over a larger set of distances.
2. Calibrate the RGB camera by using the calibration tool. Use the RGB video stream.
3. Calibrate the IR camera by using the calibration tool. Stream from the IR camera this time. For this procedure, it is recommended to cover the IR

[1]ROS calibration guide: http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration



(a) RGB camera calibration. The camera can be calibrated when a sufficient number of samples have been obtained.

(b) IR camera calibration. The chessboard pattern will be difficult to detect, because the IR projector is not blocked.

Figure 4.3: Depth camera calibration.

projector, because the IR speckle pattern makes it difficult to detect the chessboard (figure 4.3b).

The calibration program needs to know the number of inner corners of the chessboard pattern, and the size of the squares. The chessboard used in this project has 6 by 9 inner corners and the size of the squares is $0.0275m$. Larger pattern squares will make it easier to detect the chessboard pattern over a larger range of distances.

4.2.3 Power Supply and Battery Safety

A possible power supply setup is shown in figure 4.4.

The car battery is now placed in the lower shelf in the rear compartment where it is surrounded by aluminium. This presents an unacceptable risk, because a short circuit is much more likely if the battery poles are uncovered. To reduce the possibility of having a short circuit, a rubber pad was glued to the inner wall and roof next to the battery in the rear compartment. New isolated battery connections provide additional protection.

4.3 ROS Integration Overview

Implementation procedure for mobile robot:

- Decide on ROS message interface.
- Write interfaces for the motor drivers.

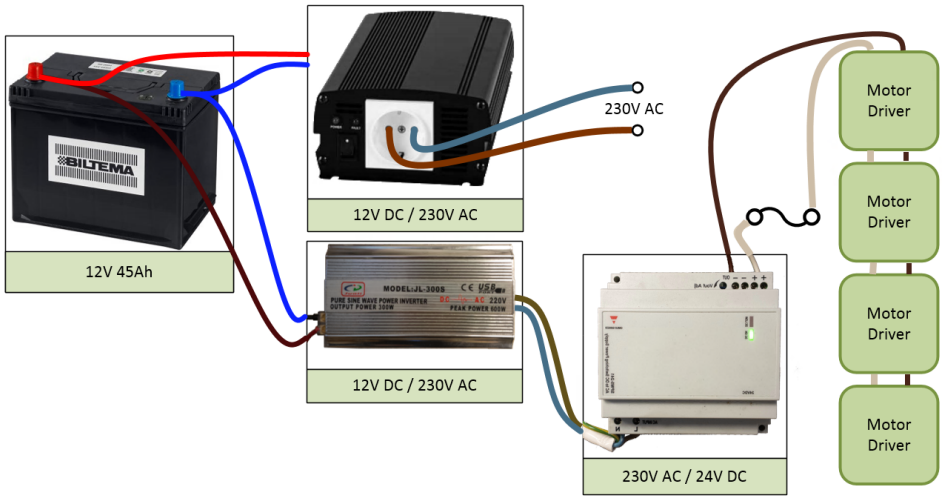


Figure 4.4: Example of a feasible power supply setup.

- Create a description of the physical structure and properties of the robot in URDF.
- Extend the model to enable simulation in Gazebo.
- Publish coordinate transform data via *tf* and visualize it in `rviz`.
- Add sensors, with driver and simulation support.
- Apply algorithms for navigation and other functionality.

The robot software implementation is placed within a `catkin` workspace (see section 3.4.2) that contains all the project specific files that are necessary to build and run the robot system. The overarching file system is shown in figure 4.5. Note that there are several references to the Three Letter Acronym (TLA) ”mar”, which is short for Mobile Autonomous Robot (MAR).

Each package will be introduced in the following sections. The purpose of presenting the file systems is to clarify which parts of the system that was implemented by this author. A short description of each package is given in table 4.1.

4.4 Modeling

A robot model will serve two purposes in this implementation. First of all, the system needs a definition of how the sensor inputs are placed with respect to the

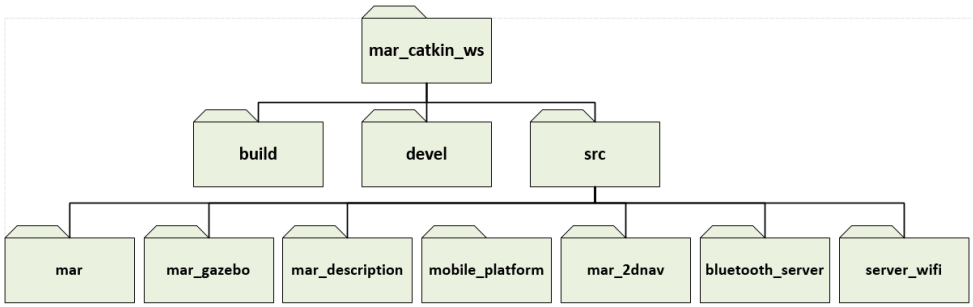


Figure 4.5: Overarching file system. The ROS packages are located within `src`.

Packages within `mar_catkin_workspace`	
`mar`	Launch files for the real robot hardware and for RTAB-Map.
`mar_gazebo`	Launch files for the simulated robot and for RTAB-Map.
`mar_description`	URDF files for both the simulated and the real robot.
`mobile_platform`	Programs for handling and processing velocity commands. One such node serves as an interface to the motor control card.
`mar_2dnav`	Configuration and launch files for both the real and simulated robot.
`bluetooth_server`	A node that serves as a Bluetooth server based on the Qt5 API.
`server_wifi`	Contains the node responsible for communicating with the OCS.

Table 4.1: List of custom made packages.

`base_link`. The origin of `base_link` is associated with a coordinate frame. This frame, and any other frame in ROS, is right handed, i.e. the positive x direction is *forwards*, positive y points *left*, and positive z is *up*. The robot pose will be based on the transformation between the world and the `base_link` frame.

The second purpose of the model is simulation. Being able to simulate the robot system has been invaluable throughout this project. The robot model is represented in an XML-based modeling language called URDF (Unified Robot Description Format). There are two *.urdf* files within the package `mar_description`; one for the simulator and one for the real robot hardware. The following sections presents how these models were defined.

4.4.1 Physical Dimensions

Step one in building the model was to define its geometrical shape. The current model shape consists of several links. Each link is defined as a shape and a size. The links are connected together by joints that define the coordinate transformation between the links. All links were modeled as either cuboids or cylinders, in order to simplify and speed up the modeling process. All joints are static except for the wheels which are continuous joints. For simplicity, the robot arm is modeled by a dummy link with the shape of a cylinder.

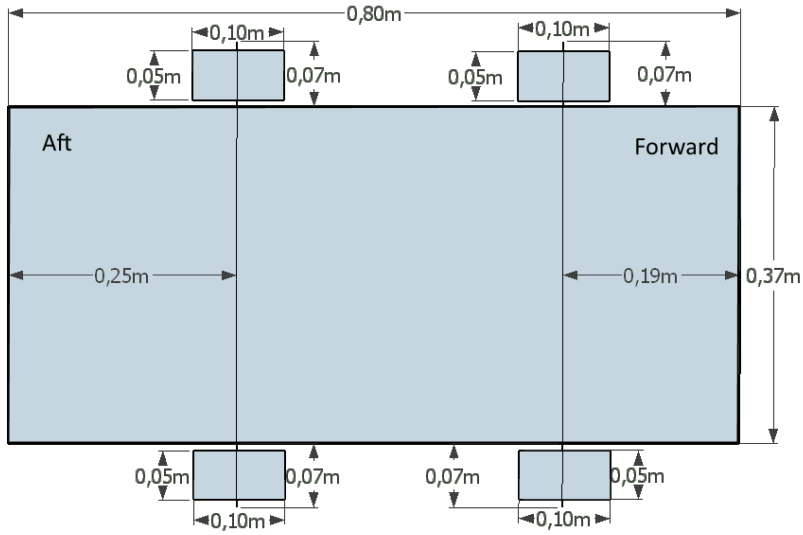


Figure 4.6: The robot footprint. Dimensions are used for the navigation planners and for modeling.

After defining the model shape, it is time to add some additional physical attributes to each link. Each link requires an inertia tensor in order to simulate the model. It is also useful to define a collision volume for each link. In this model, the collision volume is equal to the geometric shape of the link without exceptions. Inertia tensors for each shape is based on equations 4.2 or 4.3.

The inertia tensor:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{4.1}$$

Inertia tensor for a solid, uniform cylinder where the radius r is measured in parallel to the $x - y$ plane, and h is parallel to the z axis:

$$I_{cylinder} = \frac{1}{12} m \begin{bmatrix} (3r^2 + h^2) & 0 & 0 \\ 0 & (3r^2 + h^2) & 0 \\ 0 & 0 & 6r^2 \end{bmatrix} \tag{4.2}$$

Inertia tensor for a solid, uniform cuboid. The subscript of l indicates which axis l is measured along:

$$I_{cuboid} = \frac{1}{12} m \begin{bmatrix} (l_y^2 + l_z^2) & 0 & 0 \\ 0 & (l_x^2 + l_z^2) & 0 \\ 0 & 0 & (l_x^2 + l_y^2) \end{bmatrix} \tag{4.3}$$

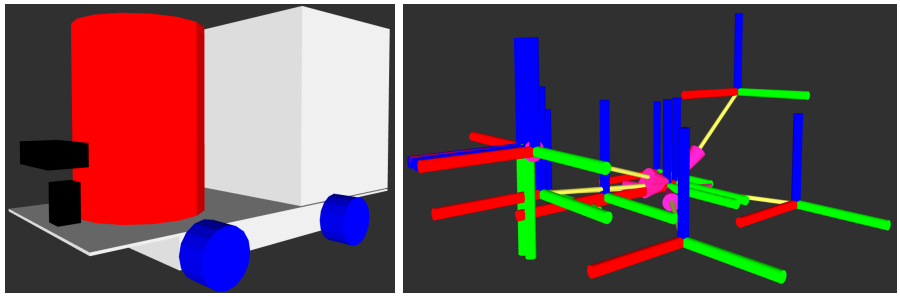
The mass of each link is guesstimated. Consider the `base_link` as an example. The base link was measured to be 5 mm thick, 37 cm wide and 80 cm long. Assuming that the density of aluminium[2] is $2.7 g/cm^3$, the mass of this link was calculated to be $\approx 4kg$. The base link was defined as follows:

```
<link name="base_link">
  <visual>
    <geometry>
      <box size="0.8 0.37 0.005"/>
    </geometry>
    <material name="silver"/>
  </visual>

  <collision>
    <geometry>
      <box size="0.8 0.37 0.005"/>
    </geometry>
  </collision>

  <inertial>
    <mass value="4" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.04564" ixy="0.0" ixz="0.0"
      iyy="0.21334" iyz="0.0"
      izz="0.25879" />
```

[2] https://en.wikipedia.org/wiki/Aluminium



(a) Complete URDF model when viewed in `rviz`.

(b) URDF model in `rviz` with all link frames and transformations.

Figure 4.7

```
  </inertial>
</link>
```

4.4.2 Connecting the Links

Robot links are connected by joints. A joint defines a translation and rotation from the coordinate frame of a parent link to that of a child link. Each joint has two attributes: *name*, for example "base_link_to_left_wheel" and type, for example "prismatic" or "continuous". All joints in the mobile robot are static, except for the wheel joints which are continuous. Correct joint transformations is very important when placing the sensors (figure 4.8). A discrepancy between the real sensor-to-base transform and the modeled transform will cause misaligned sensor input.

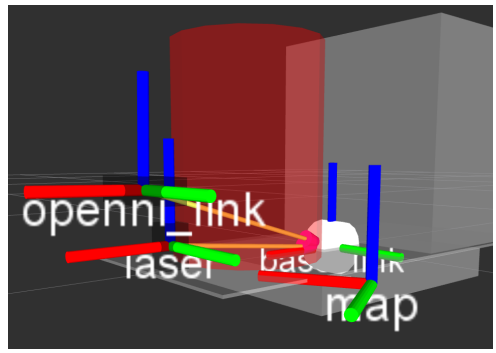


Figure 4.8: Robot model with frames for laser, Kinect, robot base and map.

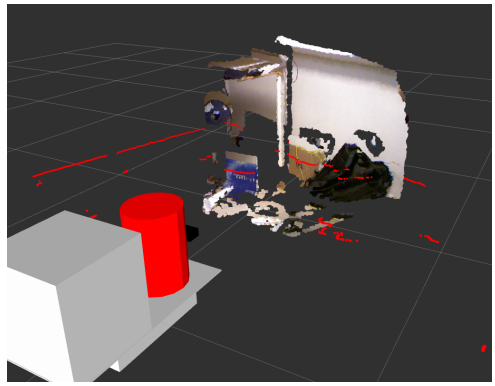


Figure 4.9: Sensor input placed with correct transformations from `base_link`.

4.5 Simulations

Robot simulation was done in Gazebo; a simulation tool with good interfaces to ROS. The same ROS graph was used for both the simulated and real version of the robot, except from the sensors and actuators, and some minor parameter changes.

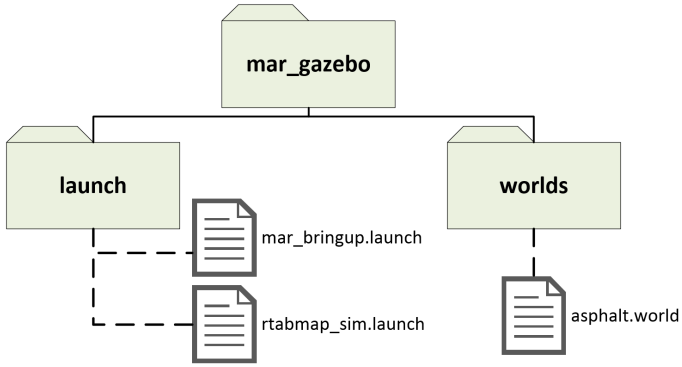


Figure 4.10: Folders and files specific for the simulator.

To properly simulate the robot, Gazebo needs a way to simulate the differential drive and the sensors in addition to the physical description of the robot. An expanded URDF description of the robot, `mar_model_sim.urdf`, intended for Gazebo was placed within the `mar_description` package (figure 4.5).

4.5.1 Robot Description Plugins

The URDF model was expanded with plugins that simulate the motor control card, the LIDAR and the Kinect. The plugins are provided by Gazebo and are intended for use in ROS.

The motor controller is simulated by the `skid_steer_drive_controller` plugin. This plugin was preferred over the `differential_drive_controller` because it supports four wheel joints instead of two, and because the skid steering controller was considered to be good enough.

Attributes for gazebo's sensor plugins are based on the technical specifications[hok] for the real sensors.

4.6 Motion Control

4.7 ROS Nodes for Motion Control

4.7.1 Velocity Command Sources

There are four ways to control the robot:

/cmd_vel_loc Local keyboard input.

/cmd_vel_wifi Wireless teleoperation from the OCS.

/cmd_vel_bt Wireless teleoperation from a hand-held Bluetooth device.

/cmd_vel Commands from the navigation stack in ROS. `/cmd_vel` is the default velocity command topic for the navigation stack.

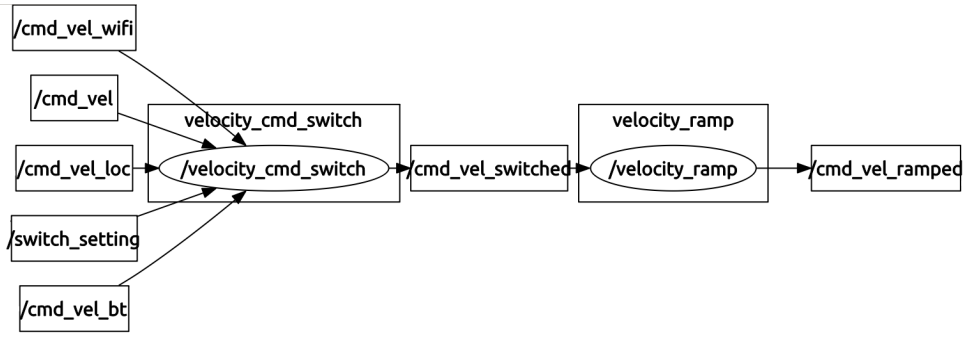


Figure 4.11: Nodes and topics for motion control. (see figure 3.4 for an explanation of this figure)

4.7.2 Motor Control Card Firmware on XMEGA A3BU

XMEGA A3BU is an evaluation board developed by Atmel. The The implementation presented here is an adaptation of Petter Aspunviks implementation [Asp13]. The following paragraphs presents the most significant firmware changes that were made.

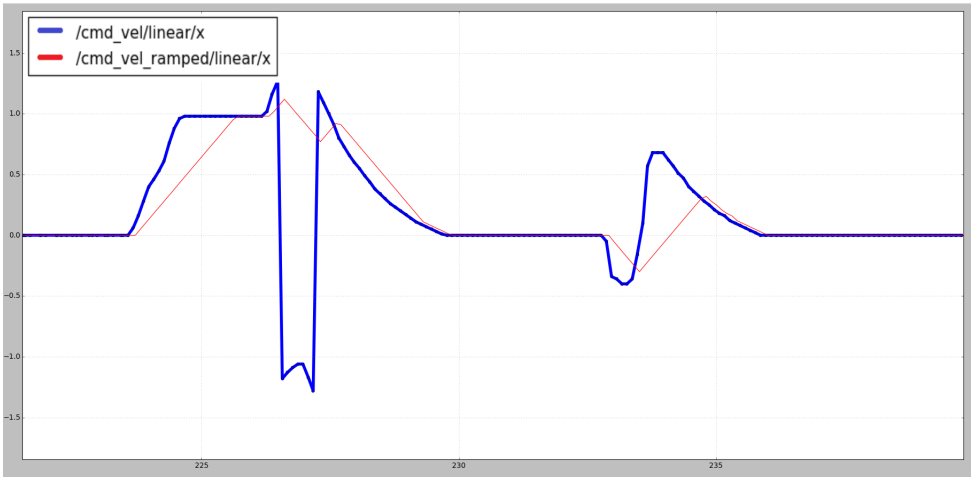


Figure 4.12: The node `velocity_ramp` limits the rate of change of the velocity command sent to the motor control card. The blue line represents commands entering `velocity_ramp`, while the red line shows the acceleration constrained output command.

The firmware will now receive angular and linear velocity commands based on the `geometry_msgs/Twist` message format in ROS, and translate these into the command format used by each motor. Speed settings for each motor is based on Pulse Width Modulation (PWM).

There were two requirements for this implementation:

1. When velocity commands from the operating system are either absent or incomplete, the robot shall stop.
2. The program shall translate linear and angular velocity commands into wheel commands.

The connections in figure 4.13 are the same as in [Asp13], except for the installation of more secure connections under the robot. The old connections were insecure, and the risk of short circuits was substantial.

ROS-Motor Driver Communication

From Velocity Commands to Wheel Commands

Within ROS, velocity commands are passed around between nodes in the form of the message `geometry_msgs/Twist`. This message type can be viewed as a *struct*

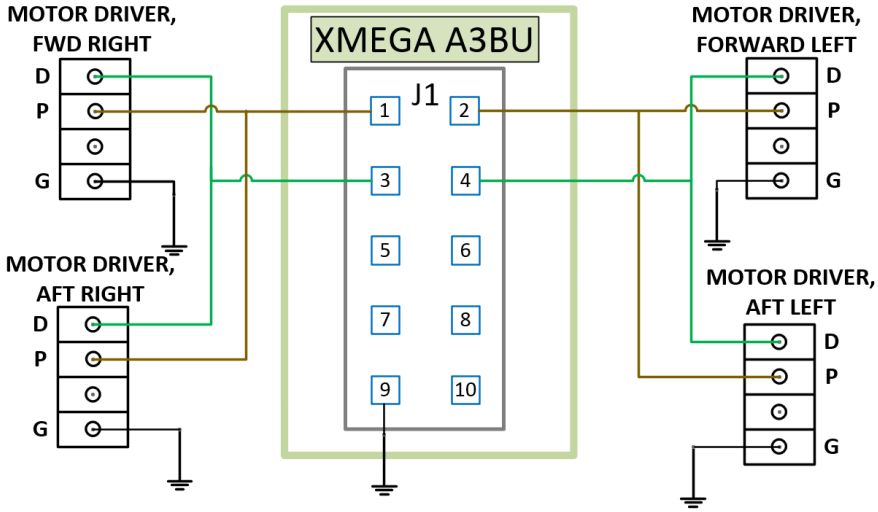


Figure 4.13: Connections between each wheel motor driver and the motor control card, XMEGA A3BU. The connections are unchanged from [Asp13], except for some improved connection for better short circuit prevention.

with the following contents:

```
Vector3  linear
Vector3  angular
```

where each vector contains float values for the directions x, y and z with respect to the robot's base frame. Because of the motion constraints of this robot, only `linear.x` and `angular.z` are of relevance, and the data which is passed to the motor control card (XMEGA A3BU) is therefore limited to these two values. The motor control card must now translate the linear and angular velocities into wheel speeds. Next, these speeds must be related to a duty cycle for the PWM signal which controls each of the four motors.

To perform the translation, it is assumed that the mobile base can be described as a vehicle with differential drive steering. Wheel commands will only distinguish between left or right - not front or aft. Equations of motion which relates angular and linear velocity to wheel velocities can be found in[Coo11].

$$\omega = \dot{\psi} \tag{4.4a}$$
$$v_{left} = \omega(R - W/2) \tag{4.4b}$$
$$v_{right} = \omega(R + W/2) \tag{4.4c}$$

W is the spacing between the wheels as shown in figure 4.15. In [Coo11], the parameter R represents the instantaneous radius of curvature of the robot trajectory. This mouthful will be substituted by the linear velocity v in the following equations, because $v = R\omega$ (similar to the linear speed of a wheel). This yields two equations for the wheel speeds, v_{left} and v_{right}, based on angular and linear velocity, w and v.

$$v = R\omega \tag{4.5a}$$
$$v_{left} = \frac{2v - \omega W}{2} \tag{4.5b}$$
$$v_{right} = \frac{2v - \omega W}{2} \tag{4.5c}$$

4.8 Operator Control Station (OCS)

The OCS allows an operator to control and monitor the robot through a graphical user interface. `MainWindow`

4.8.1 Graphical User Interface

A Qt-based Graphical User Interface (GUI)...

4.9 The Hand Held Remote Control - *Robot Leash*

Because the OCS is only partially implemented, an operator will not have access to all the features on the robot. In addition, as a safety precaution a person should be close to the robot at all times, and be ready to pull the plug. Furthermore, it is hard to control a moving robot through the on-board keyboard. These problems were countered by the Android-based remote control, *Robot Leash*.

4.9.1 Connecting to the Robot

1. The first screen after scanning for devices. There is no device filtering, and the user can select any device, but only connect through a specific service.

2. After selecting a device which provides the correct service, the user will be prompted to pair the devices.

3. The smartphone and the robot is now paired, and velocity commands from the blue control stick are passed to the robot via Bluetooth.

http://developer.samsung.com/technical-doc/view.do?v=T000000117

4.10 Mapping - Setting Up RTAB-Map

This robot is using Real-Time Appearence Based Mapping (RTAB-Map) for SLAM. As mentioned in section 3.9.3, RTAB-Map itself has been developed over the last decade by IntRoLab at Université de Sherbrooke in Canada. This section presents how RTAB-Map was configured for this robot.

4.10.1 Configuration

As all ROS programs which are a part of the robot system, RTAB-Map will run as a node that subscribes and publishes topics. The first task in configuring RTAB-Map is to connect the robots sensor data to the RTAB-Map node. The node, called `rtabmap`, can build 2D occupancy grids and/or 3D point cloud representations of the environment. In this project, RTAB-Map is configured to do both. The configuration is based on a guide[rta] provided by the developers. To perform SLAM, the mapping node subscribes to odometry, 2D laser scans and camera information. There are five possible sensor configurations with the Kinect[rta]:

1 - Kinect + LIDAR + Odometry

Sensor data can be sent directly to `rtabmap`.

2 - Kinect + Odometry + Fake 2D laser from Kinect

2D laser scans are generated by passing depth images from the Kinect through the node `depthimage_to_laserscan`.

3 - Kinect + LIDAR

This is the configuration that was used in this project. Odometry data is generated by a the scanmatcher within Hector SLAM (section 3.9.2). `rtabmap` receives point clouds

4 - Kinect + Odometry

This configuration is suitable for uneven surfaces, i.e. when the vehicle is not constrained to a plane. Supports *roll*, *pitch* and *yaw* rotations.

5 - Kinect

In this mode, odometry will be generated by the `rgbd_odometry` node bundled with the `rtabmap_ros` package. This node publish odometry messages based on feature correspondences in consecutive RGBD images received from the camera.

4.10.2 Adding 3D Obstruction Detection

`rtabmap_ros` contains a 3D obstacle detector in addition to the mapping node `rtabmap`.

4.11 Navigation

4.11.1 Global Path Planning

4.11.2 Local Path Planning

Obstruction Detection

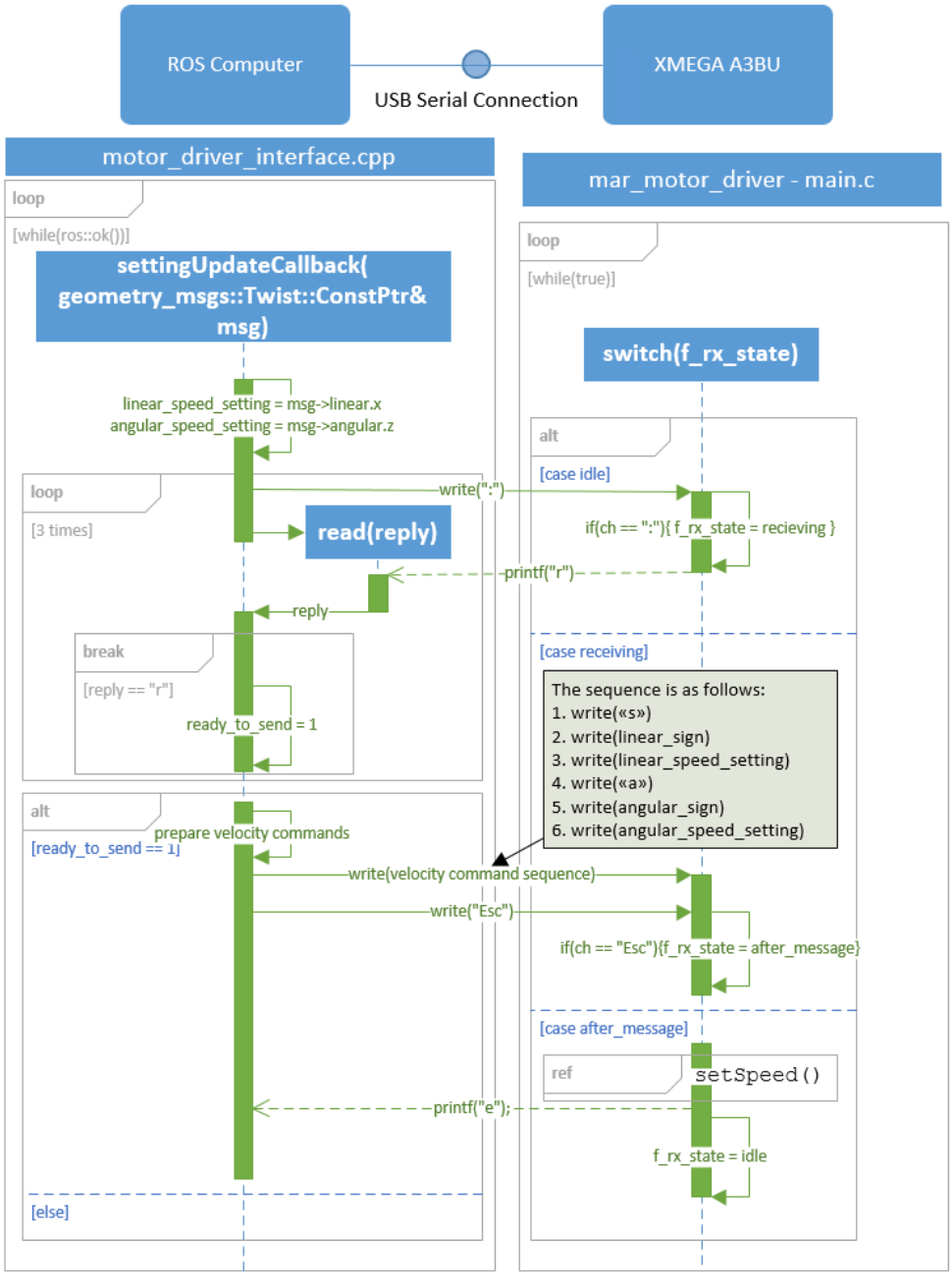
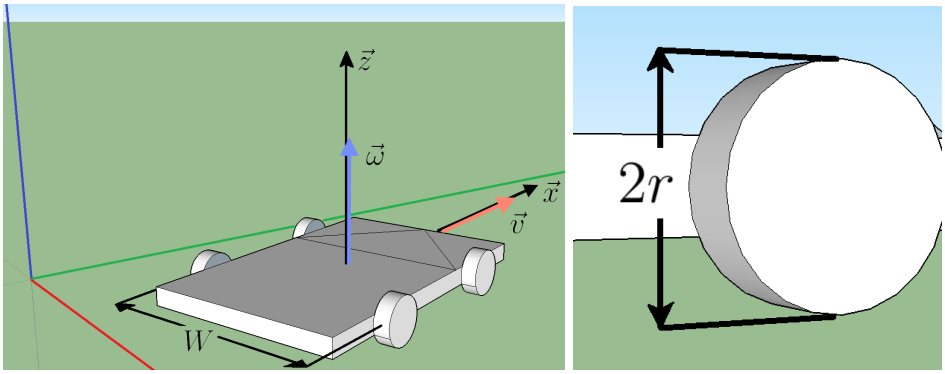
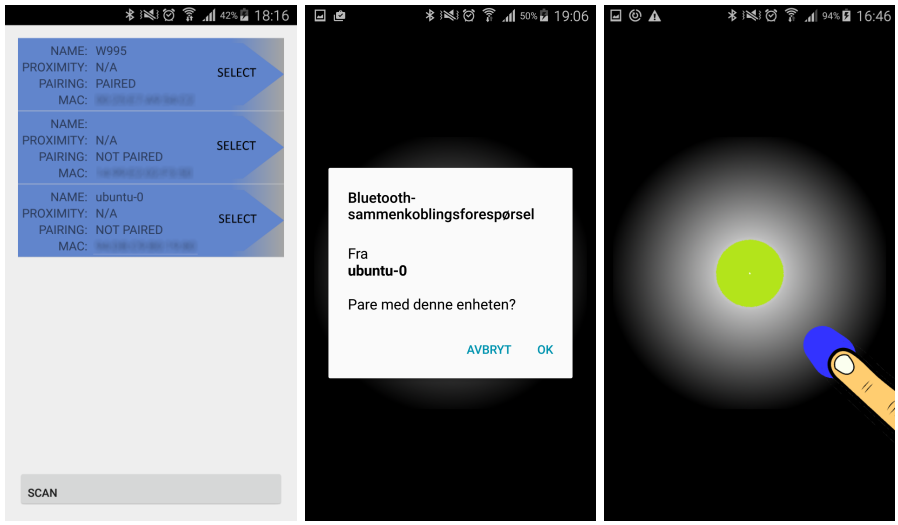


Figure 4.14: Velocity command transmission sequence from the `motor_driver_interface` in the ROS computer to the motor control card (XMEGA A3BU).



(a) Differential drive parameters. (b) Wheel diameter.

Figure 4.15: Parameters for differential drive kinematics. Note that the frame vectors $\vec{z}$ and $\vec{x}$ refer to the base frame of the robot in this case, and not the world frame.



(a) First activity with device list. (b) The user is prompted to pair with the robot. (c) Controlling the robot with the stick.

Figure 4.16: A typical use case for ”Robot Leash”.

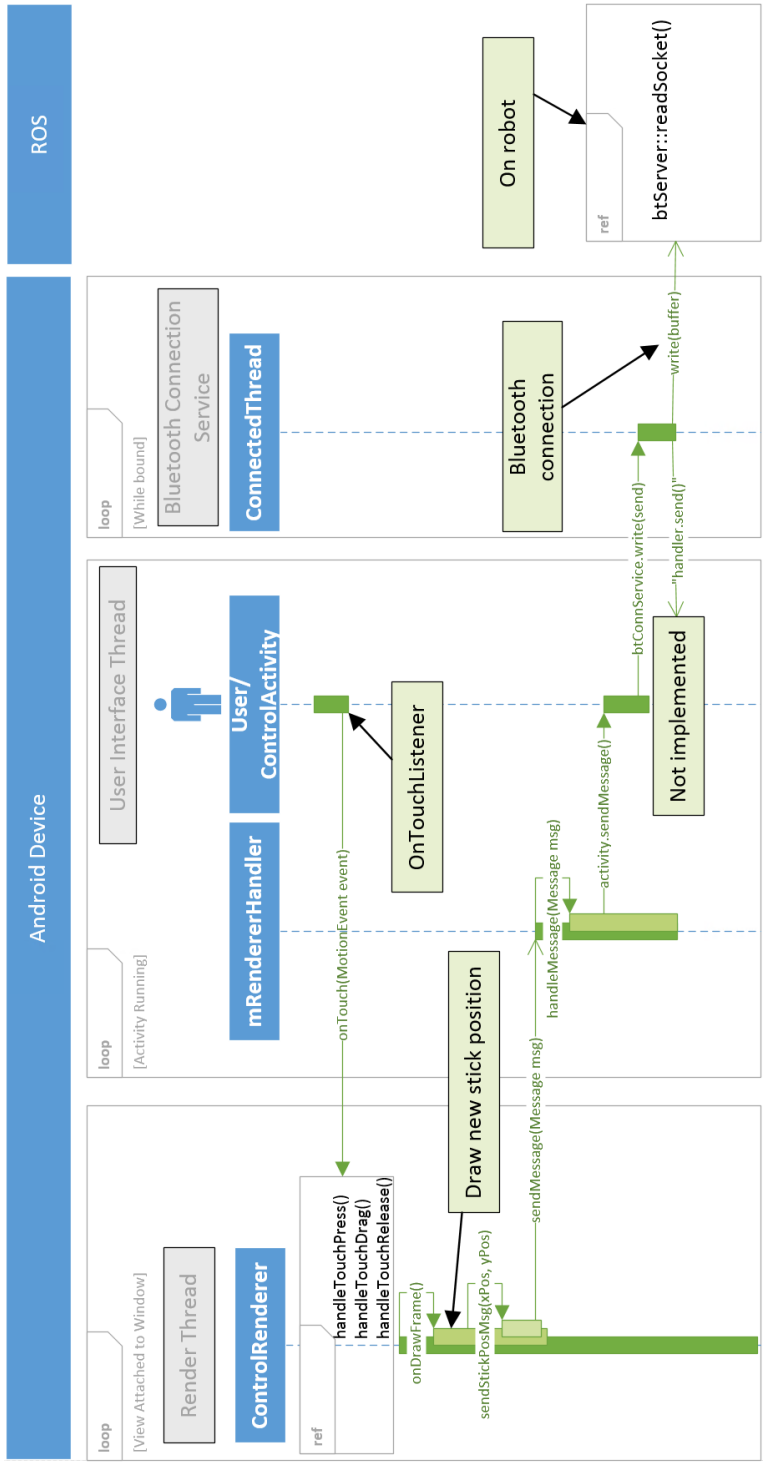


Figure 4.17: Sequence diagram illustrating how user touch gestures are detected and propagated through the application, before being transmitted as commands to the robot.

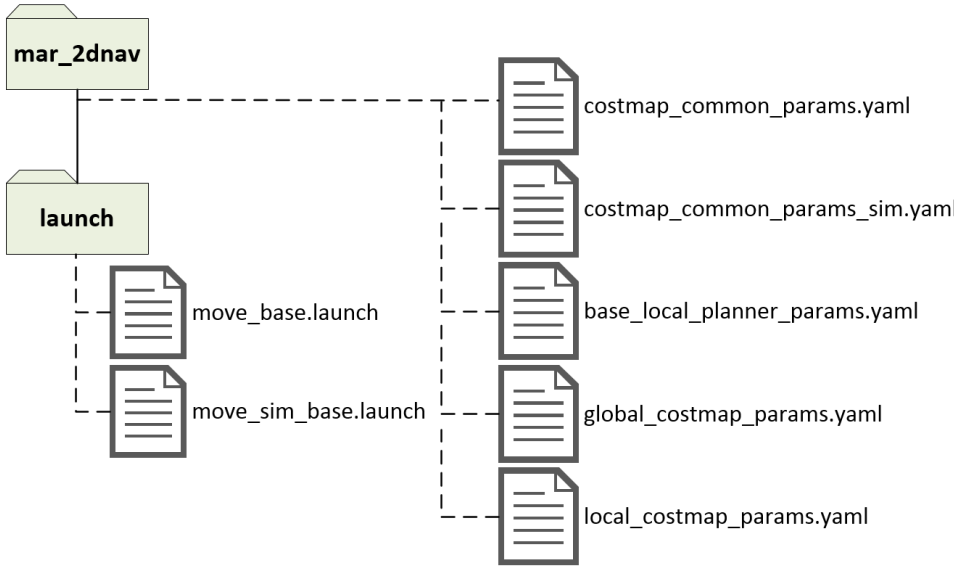
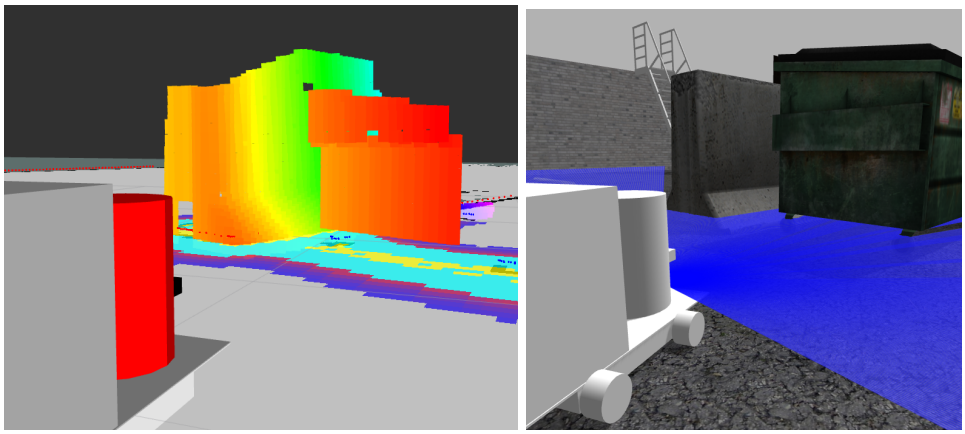


Figure 4.18: Files for configuring and launching the navigation stack.



(a) A point cloud representation of the obstruction. Notice how the local costmap is based on the detected point cloud.

(b) The obstruction in the Gazebo simulator. Notice how the LIDAR only detects the wheels below the container.

Figure 4.19: Detecting obstructions in 3d.

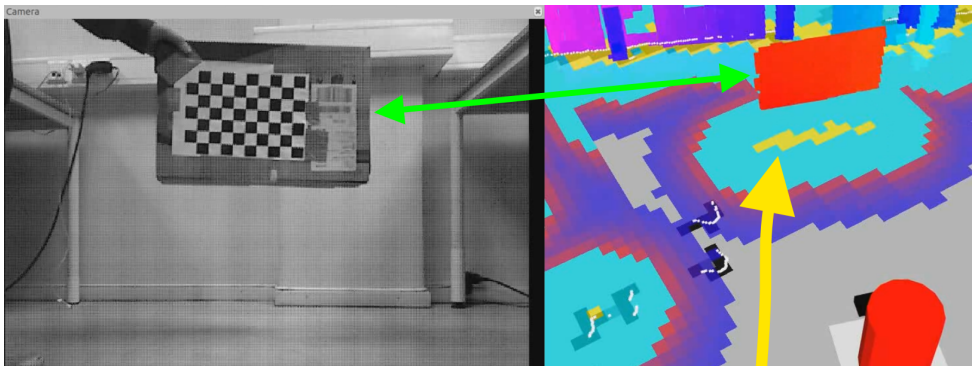


Figure 4.20: 3D Obstacle detection with the live robot. The nodelet `obstacles_detection` filters out the floor and publishes a point cloud which can be sent to the `move_base` node. The yellow arrow points to the local cost map, which is based on real-time sensor data and used by the local planner.

Chapter 5
Results

5.1 Introduction

This chapter presents how the robot and the supporting implementations were tested and the results that where obtained. The same software system was used for both the simulated and live robot. It was still necessary to have some separate launch and configuration files for the simulated and hardware version (section 4.3).

5.2 Testplan

The following tests will be carried out in the simulator, as well as in the real world. They will mainly focus on the navigation stack and RTAB-Map.

5.3 Simulation Results

The system was tested on a simulated model of the robot in Gazebo. A simulated environment, `Asphalt.world` shown in figure 5.1, was populated with objects in

Supporting Functionality

Evaluate	Description
Mobile application, "Robot Leash"	Use the mobile application to manually steer the robot.
Operator Control Station	Steer the robot from the OCS while monitoring the robot through the live video stream.
Motor controller on XMEGA A3BU	Verify ability to command the wheels. Confirm that the vehicle stops when velocity commands from ROS are absent.

Table 5.1

Core Functionality

Evaluate	Description
Multi Session Mapping	Verify that the robot can rediscover areas which have been mapped in a previous mapping session.
Loop Closure Detection	As a core functionality in RTAB-Map, it is critical to evaluate the loop closure mechanism.
Autonomous Navigation	Perform a set of tests on the navigation stack. The tests should evaluate path planning with moving obstacles. Different parameters should be tested and evaluated. Observe how the robot handles narrow passages. Evaluate robustness of the navigation stack for this robot.

Table 5.2

order to provide a test environment with distinctive visual features for the visual mapping approach, and obstacles for the navigation stack.

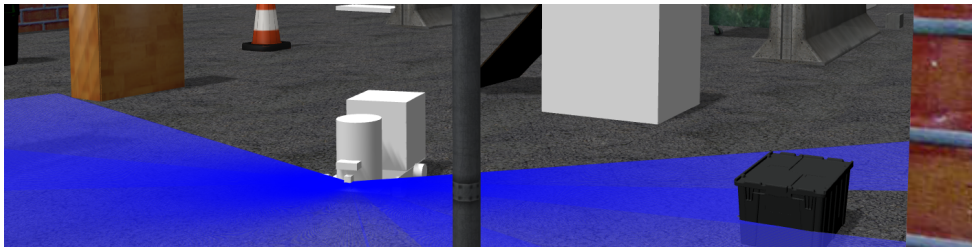


Figure 5.1: The ”Asphalt” world in Gazebo.

5.3.1 Mapping

RTAB-Map on the robot in Gazebo allowed controlled testing of edge cases in a controlled environment. The ”Asphalt” world proved to be a challenge for RTAB-Map - at least with the parameters that were used during testing. Figure 5.2 shows an example of a resulting 3D map.

The map quality varied greatly between the mapping trials. The path of the robot was found to have a significant impact on the recorded path between the stored locations in the RTAB-Map system. An example of a problematic path is to drive the robot in parallel to a wall. Such a path creates few or none distinctive features as long as the robot follows this path. Figure 5.3 shows two problematic events. First, a loop closure is detected based on similarities detected on the asphalt plane. Because

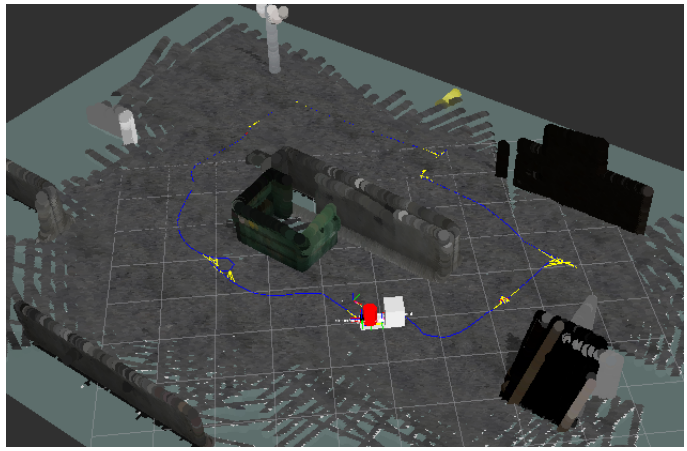


Figure 5.2: An example of a resulting point cloud map after running RTAB-Map in Gazebo.

the loop closure is wrong, a wrong pose transform correction will be propagated backwards along the path of the robot, ultimately resulting in a displaced map.

Another error occurred during a multi session mapping trial. The robot detected a wrong loop closure at a location with similar features to a previously mapped location. This resulted in the map seen in figure 5.5. Other localization problems were apparent when driving in open areas. The estimated location of the robot would fluctuate as when turning the robot as features passed in or out of view.

5.3.2 Autonomous Navigation

Testing the navigation stack in Gazebo fulfilled two goals. The first goal was to learn how the system behaved with different parameters and to uncover potential problems with the system before attempting to test the live robot. The second goal was to evaluate the ability to relocate the robot and avoid obstacles.

The for the first trial, the robot footprint was extended well beyond the physical mobile base. The purpose of this was to prevent obstacles from getting too close to the Kinect and LIDAR. As the minimum detectable range for the Kinect is $0.5m$, the footprint was extended $0.5m$ (measured minimum depth) beyond the front of the mobile base. A second parameter to be tuned is the obstacle inflation radius, i.e. the radius beyond each obstacle that is expensive or impossible to traverse. Figure 5.6 illustrates both the big footprint and the obstacle inflation radius.

During trials with the big footprint, the robot showed good collision avoidance but an aversion to narrow passages. Setting the obstacle inflation radius is also a dilemma in

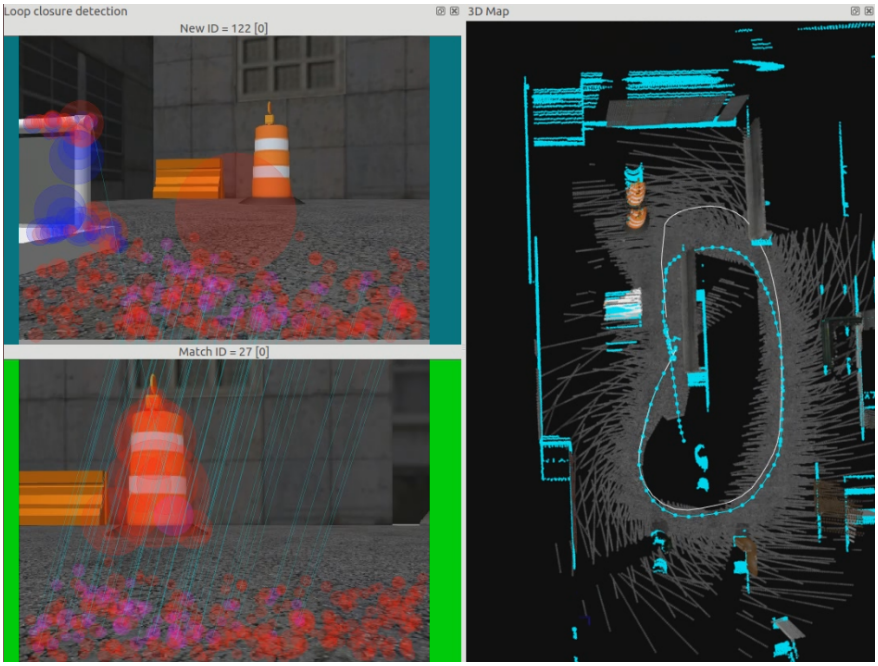


Figure 5.3: Example of an incorrect loop closure detection. The pink circles indicate matching features. The right part shows an incorrect map adjustment. Observe how the matching features are located on the asphalt plane.

choosing between large margins for the global path or the ability to navigate through narrow passages.

In later navigation sessions, the robot footprint was reduced to a size slightly larger than the physical robot base. During some of the testing sessions the robot would get stuck in the obstacles.

Sometimes, the robot would never actually reach the goal state, but rather circle around it.

5.3.3 Live Robot Results

Due to time constraints, it was no time to tune the parameters of RTAB-Map. RTAB-Map is therefore used with the default parameters.

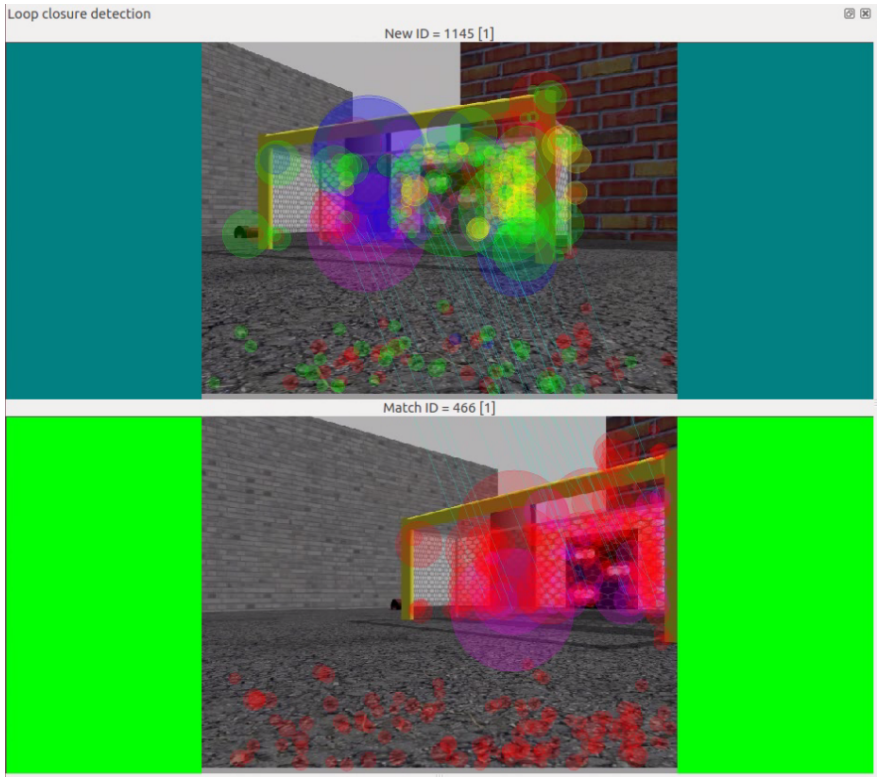


Figure 5.4: An example of an accepted and correct loop closure hypothesis. This example is from the "Asphalt" world simulated in Gazebo.

Safety Features

Loop Closure Detection

Loop closure detection was carried out on the first floor of Gamle Elektro at Gløshaugen, NTNU (figure 5.7a). This environment provides a good mix of featureless and feature rich surroundings. It will also provide an additional challenge because of the many students that use the hallways. Most importantly, there are loops in the environment that allow testing of vision based loop closures and odometry error correction.

The first test run revealed two problems with the implementation, the first being odometry errors and the second being a failure to visually detect loop closures. Odometry based on laser scans would wrongfully indicate a change in the robot heading in some cases, and in other cases fail to correctly indicate heading changes when rounding corners.

Figure 5.5: An example of incorrect map merging. This case occurred in the ”Asphalt” world simulated in Gazebo.

In later experiments it was found that different mapping techniques and path choices could either prevent or cause odometry errors. It was also found that it is helpful to start a mapping session in an area that is rich in distinctive features. The map and floor plan comparison shown in figure 5.7 shows a map where a loop closure was successfully detected. In the same figure, notice how the upper hallway is misaligned to the rest of the map. The robot failed to detect any good visual features in this area. Figure 5.8 shows the resulting point cloud map of the same area.

Multi Session Mapping

Multi session mapping was tested by performing an initial mapping run and then starting a second mapping session from an unknown location. All multi session mapping trials were successful.

Navigating Among Static Obstacles

The first live navigation trials were performed by setting up a static obstacle course in a hallway. Only a few people were using this hallway, so multiple trials could be carried in under similar conditions. Most navigation trials were carried out in mapped areas.

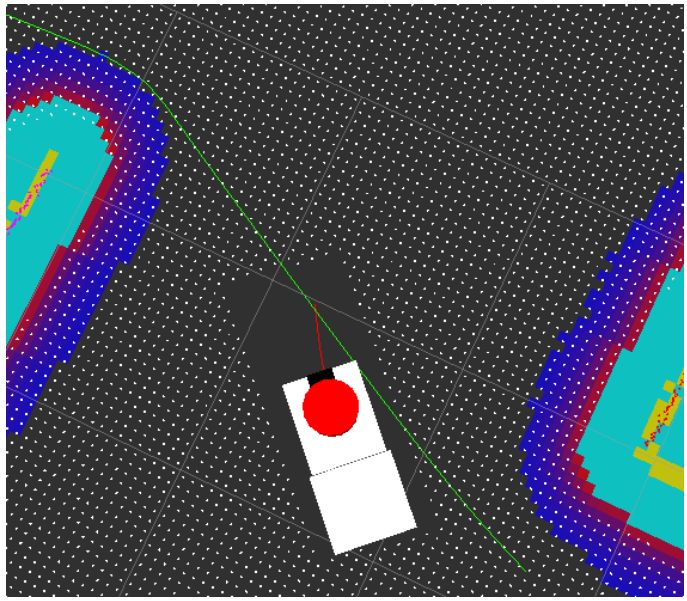
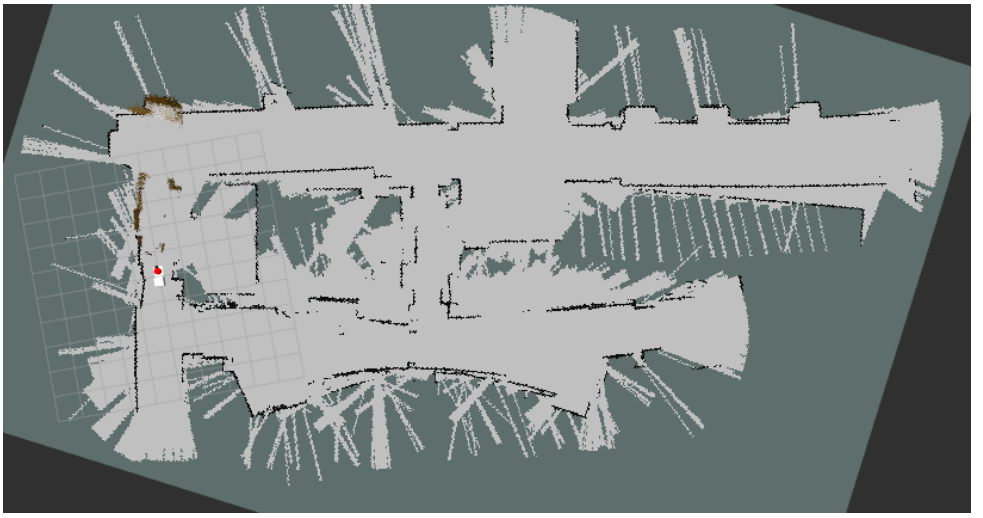


Figure 5.6: The robot footprint is illustrated by the clear rectangle that surrounds the robot model. The coloured areas are map locations with high cost.

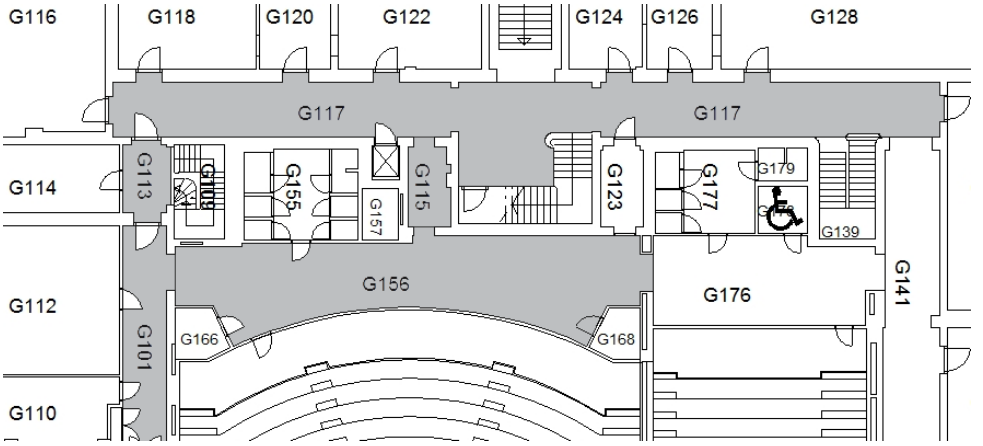
Avoiding Moving Obstacles

The navigation stack is configured to use a static global map for global planning and a local cost map that is linked with the mobile base and is based on real time sensor data. The local cost map will receive LIDAR laser scans, Kinect point clouds and point clouds representing detected obstacles detected by `rtabmap`. Being a real-time map, the local cost map should enable the local planner to avoid people and other non-static obstacles.

Moving obstacle avoidance testing was performed by having a person move into the planned path of the robot at different distances. Other avoidance situations would occur randomly as people were walking by the robot in the hallways. Tests showed that the robot is able to avoid non-static obstacles if the obstacle is observed at a distance larger than $0.5 - 0.8m$. Figure 5.11 shows that the robot has successfully planned a new path around a person. If an obstacle appeared any closer than this, the new circumnavigating plan would either be too close to the original plan or not be planned at all. Detection and planning was not instantaneous. Some time would pass before the obstacle was detected and a new local plan was generated. This detection delay reduced the detectability of people walking by the robot. Figure 5.10 shows an example of when the local cost map was lagging behind the actual moving obstacle.



(a) Resulting occupancy grid after a mapping session. The mapping method is struggling with the hallway in the upper part of the image.



(b) Floor plan of Gamle Elektro, first floor.

Figure 5.7: Comparison between mapped occupancy grid and floor plan.

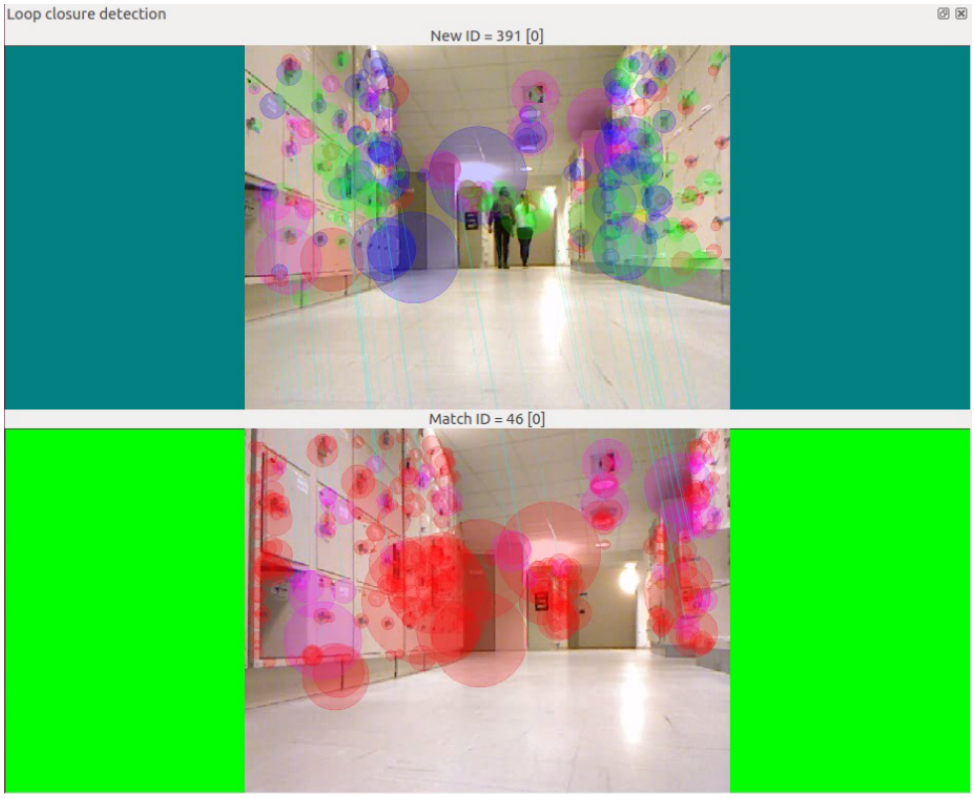


Figure 5.8: An example of an accepted loop closure hypothesis. This example is from the "Asphalt" world simulated in Gazebo.

5.4 Discussion

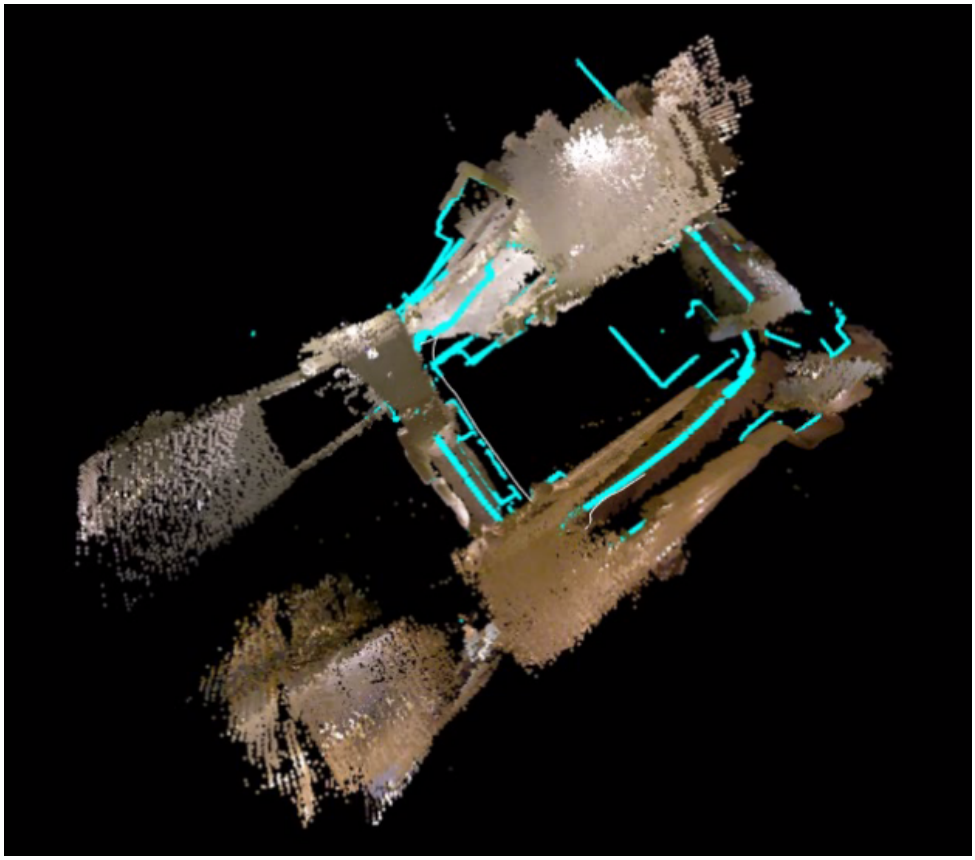


Figure 5.9: The resulting 3D map of the same area as in figure 5.7a.

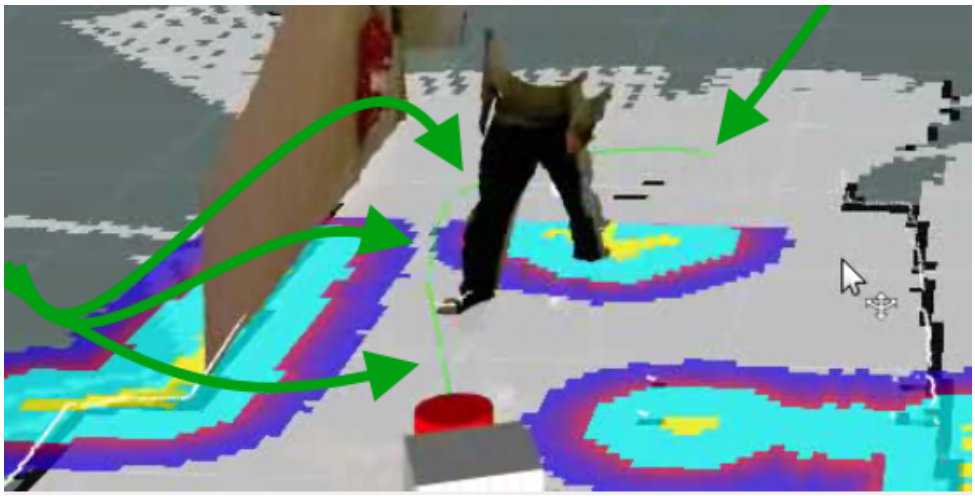
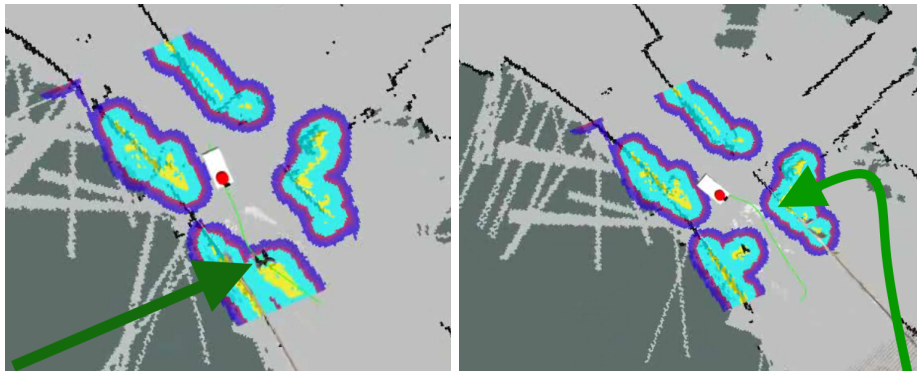


Figure 5.10: Avoiding moving obstacles with a new plan that circumnavigates the detected obstruction. In this situation, the obstacle was moving too fast for the local planner. The right leg is not yet registered as an obstacle.



(a) A person has moved into the path of the robot.

(b) A new path is planned, avoiding the new obstacle.

Figure 5.11: Moving obstacle avoidance. The local cost map, shown as coloured spots on the occupancy grid, is based on real-time sensor data.

Chapter 6
Discussion

6.1 Introduction

So far the report has presented a large number of topics and implementations.

This chapter will seek to assess the implementations presented in chapter 4 based on their performance recorded in chapter **??**. The following discussions will include qualitative evaluation of suitability for offshore robotic maintenance.

This chapter will examine the results presented in chapter 5 and discuss them in light of the initial problem description for this thesis. The first section will assess the extent to which the project covers the problem description. Then the potential strengths and weaknesses of each subsystem.

6.2 Task Fulfilment

6.3 Overall Assessment

The overall system, as it is at the end of this master's project, is a functioning proof of concept for a mobile autonomous robot. All planned modules were implemented. Some of these, however, are only capable of demonstrating basic functionality and the possibility of more robust and complete implementations. The most important features, camera based mapping and navigation, were successfully implemented and configured.

6.3.1 Choice of Development Tools

Using ROS as a development framework, might have been the most important factor that contributed to a functional solution. In the end, ROS proved to be a flexible and rich tool, despite its novel structure and initial learning curve. Experienced users of ROS will most likely be able to rapidly test and implement robot concepts. Since its

inception in 2007, ROS has become a mature and rich set of tools and functionality packages that anyone can implement and develop further. The node structure is also a good way of structuring the entire system into self contained and reusable modules. This makes it easier to reuse parts of this implementation in later projects, and will hopefully benefit ensuing projects on this topic.

6.3.2 Assessment of Prototype Design

Short

Current Design and Robotic Maintenance

So far, the robot prototype has been developed for a few specific functionalities. A comprehensive and long-term approach towards designing a maintenance robot has not been a priority. This subsection will go through some specific design choices on the current robot, and discuss potential shortcomings.

A typical offshore installation floor will most likely be made of steel and steel gratings with many holes, gaps and sharp edges[GP08]. The current mobile base is better suited for completely even surfaces, and would benefit greatly for a more rugged set of wheels. [GP08] is also referring to a minimum size for passage ways that could serve as guidelines for later prototypes. As the motor control card is an open loop system, slopes and increased friction for whatever reason affect the speed of the mobile base. This may cause the base to slow down, speed up or even stop completely if it is driving up a slope that is too steep.

Given that the navigation stack in ROS is thoroughly tested on square or circular bases, designing a new base to be either square or circular could increase the robustness of the navigation system. A holonomic drive could also make the robot more manoeuvrable, which may be useful in tight spaces.

6.4 Success and Quality of the ROS Integration

Integrating and configuring ROS with the existing robot was the most time consuming task of this project. The current implementation is capable of autonomous navigation and long-term map building. Remote operation from the OCS and Android device is also possible.

6.5 Assessment of RTAB-Map

The mapping session results demonstrated both strengths and weaknesses in the chosen mapping method RTAB-Map. The results show that multi session mapping

works rather well. Especially in environments with a sufficient amount of detectable visual features.

6.5.1 Quality and Thoroughness of the Tests

RTAB-Map was tested in a diverse set of indoor environments as well as in the simulated "Asphalt" world in Gazebo. A significant shortcoming of these tests was the lack of testing and comparing different parameter settings for the method. Further parameter tuning and better mapping techniques could have benefited the mapping performance.

Another shortcoming is the small number of live mapping trials. Configuring and learning to use the mapping system was a time consuming process. Problems with the robot hardware and the environment itself gave rise to additional delays. The laptop running ROS and the on-board car battery had to be recharged periodically, which was a time consuming process. The live loop closure tests were carried out at times when a lot of students were moving through the hallways. These factors made it difficult to perform comparable tests, and proved to be a complicating factor for the appearance based mapping system.

A third weakness in these trials is that the system was tested with only one sensor configuration. RTAB-Map can utilize both a RGB-D camera, a LIDAR and odometers, but these tests were only carried out with the Kinect and the LIDAR.

6.5.2 Weaknesses

Appearance based loop closure detection with RTAB-Map has many confirmed and potential weaknesses that must be addressed. Figure 5.3 from a simulator session illustrates an incorrect loop closure detection with a subsequent incorrect odometry correction of the previously visited locations. The figure shows that the matched features are based on the ground plane in the simulated world. Having a feature rich ground plane could be a weakness with the simulated world, as it is not was not a good analogue to the real input. The depth map generated by the Kinect was in fact quite sparse at the ground plane. Another error that occurred during simulations is incorrect merging of two maps of the same area (figure **??**). This particular event was caused by having two very similar locations in the same area. A potential problem is that the appearance of the environment will change based on the time of day, time of year and potential wear and tear on the surroundings. How robust the feature detectors (SIFT, SURF, Oriented FAST and Rotated BRIEF (ORB) etc.) are to such changes was not investigated during this project.

6.5.3 Strengths

RTAB-Map is very feature rich. It supports many sensor configurations, including stereo cameras. The ROS wrapper makes it easy to integrate the method into an existing robot system. The developer or user has access to hundreds of parameters to tailor and fine-tune the mapping system. Object recognition and 3D obstacle detection is also useful features that could support a maintenance robot.

6.5.4 Suitability For Robotic Maintenance

The fact that RTAB-Map is suitable

RTAB-Map is a feature rich mapping system with many capabilities that can make it suitable for a remotely operated and autonomous maintenance robot.

6.5.5 Mapping

- Repairing broken maps? What to do when map is partially broken.
- Poor odometry when surrounding are in motion, or when laser features are difficult to detect. System can be fooled easily.

Mapping with RTAB-Map

Both the simulated and the live trials show that RTAB-Map is able to perform SLAM.

Mapping trials in the Gazebo simulator RTAB-Map can be tuned and configured with hundreds of parameters.

6.5.6 The Kinect Sensor

6.6 Navigation

- Rectangle base vs. square base.
- holonomic wheel.
- Open loop wheel control (stuck when friction is high.)
- Same as for mapping. Poor odometry when surrounding are in motion, or when laser features are difficult to detect. System can be fooled easily.

Integrating the ROS navigation stack into a new mobile base was in itself a fairly simple procedure. Finding a good configuration turned out to be a more complicated process. Navigation was tested on both the simulator and the live robot. It became apparent during the testing sessions that the behaviour of the simulated robot was not analogue to the real robot. Recall that the simulated robot is controlled by a slip steering plugin in Gazebo, while the real robot is closer to a differential drive vehicle. In addition, the motor control in the real robot is an open loop system. The wheel commands from ROS will normally result in a lower linear velocity and yaw rate. As a consequence of this discrepancy between the real and simulated robots, the performance assessment of navigation stack in the simulator will have a reduced weight.

It is important to emphasize that the navigation stack was implemented in its simplest form in this thesis, and the assessment will therefore be limited.

6.6.1 The Tuning Process

There are no official tuning strategies for the navigation stack in ROS besides a basic guide to give users a general idea of where to start and what to check[ROSf]. The tuning process is currently a ”change and check” process partially based on guesswork or copying similar solutions from other projects. This is far from ideal, as it is both time consuming and a hindrance for finding an optimal solution.

It should be noted that the navigation stack has been thoroughly tested on robots with square or circular bases. The highly rectangular base ($80x37cm^2$) on this robot may have been a handicap.

As with the SLAM test sessions on the live robot, the cumbersome hardware and limited battery life significantly constrained the amount of time available for testing.

Based on the test results, the live robot should have a higher minimum speed setting. If the motor control card is expanded with a speed regulator, the minimum speed may be reduced.

6.6.2 Performance

Live testing showed promising results, despite a few quirks. The live robot would reliably plan a path to a goal location and move the base to this location. Sometimes, the robot would stop a few centimetres before reaching the coal location.

6.7 Suitability for Offshore Maintenance

This is just a prototype. Mobility issues. Kinect-like sensors and ROS could be useful. It is at least an excellent tool for ”rapid” prototyping.

The current configuration has many shortcomings as a maintenance robot. In its current form, it is more suitable to test various technologies and concepts.

The current sensor configuration is set up to facilitate testing of SLAM and navigation.

The current design of the mobile base is struggling to support the weight of the robot. The small wheels are struggling with small obstacles such as door thresholds or floor gaps in the entrance to elevators. MIMROex, a comparable robot, is equipped with a variant of a differential drive base with larger wheels suitable for driving over steel gratings and slippery surfaces.

6.7.1 The Kinect

As mentioned in section **??**, the Kinect is an active sensor that measures depth by projecting an infra red speckle pattern onto the surroundings. Kinect for Xbox 360 can't be used in commercial applications because of its license. There is however similar sensors that can replace the Kinect.

6.7.2 Open Source Software and Security

ROS and other open source projects thrive on active communities of contributors. Both PCL and ROS, as well as many other libraries and frameworks, are built on a collaborative effort from researchers and developers across the globe. This is open structure is great for speeding up innovation. Issues and bugs can also be discovered more quickly by anyone. Another benefit is that every detail in an open source project is open for scrutiny by those who want to use it. This is also a problem in terms of security. While anyone can find bugs and issues, the code is also open to those who are looking for possible exploits and vulnerabilities. If a system is targeted for sabotage, and it is widely known that the system uses open source software, it might be more vulnerable to security threats.

Chapter 7
Conclusion

7.1 Future Work

7.1.1 Autonomous Non-Destructive Testing

Advancements in Artificial Intelligence (AI), big data and machine learning opens up exciting possibilities for autonomous NDT. Branches of this technology is usually encountered in the context of image recognition, i.e. teaching machines to understand what they see. The same concepts may be applied to forms of NDT besides regular visual sensor input, such as ultra sound or eddy currents for corrosion detection.

7.1.2 Large Scale Kinect Fusion - Kintinous

Kinect Fusion has great potential for augmented reality. Augmented reality is a concept which blends the real and virtual environment. This opens up opportunities to create realistic and immersive training scenarios for the operators. Unfortunately, Kinect Fusion is limited reconstructing a rather small volume depending on the resolution. By varying the resolution, volumes can at the least cover a normal office desk and at the most cover a small room [**?**].

Kintinous...

A guide on how to build Kintinous can be found at https://github.com/mp3guy/Kintinuous. The procedure is complicated, as it usually is for experimental builds. It is recommended to attempt the procedure on a fresh install of Ubuntu 14.04 or 15.04 [Kind].

Improve the Communication Protocols

Communication between ROS and the XMEGA A3BU, the Bluetooth device and the OCS, all use the same pattern: A start byte `":"`, the message with the speed setting

and a stop byte `"Esc"`. In later projects, it could be beneficial to implement a more robust and rich communication protocol with more options for remote operation.

Implement a Fully Functional Operator Control Station

At the end of this project, the OCS provided functionality for moving the robot, and displaying live video from the Kinect.

7.1.3 Hardware

Several hardware-related issues became apparent over the course of the project - especially toward the final weeks. These issues are likely the results of many disconnected projects on the same hardware.

Kinect Sensor Location

This is the first semester in which a Kinect has been used on the robot. At the moment, the sensor is placed directly over the LIDAR device. Because the depth sensor in the Kinect for XBOX 360 has a minimum range of roughly $0.5m$, it cannot detect objects within reach of the robot arm. It is recommended to find a new location further back on the robot.

Combine Stereo Cameras with Kinect-like Sensors

As mentioned, both active and passive depth cameras have limitations. The Kinect does function in direct sunlight, but it can measure depth in the dark. Passive depth sensors, for example stereo cameras, does depend on visible light to sense anything at all. While RTAB-Map does depend on visibility for loop closure detection, there are other SLAM methods, e.g. *Kinect Fusion*, which do not. An implementation could use a light sensor to sense light that may interfere with the Kinect. Light levels could be compared to a threshold and switch between the stereo cameras or the Kinect depending on how well each sensor will work in the current conditions.

On-board Computer Suitable for Moving Platforms

Because this author used his own computer to control the robot, all features related to ROS was removed from the robot at the end of the project. A new computer should be equipped with Solid State Drive (SSD) storage

Wheels

There were mainly two issues with the omni-wheels this semester: They are worn out, and one wheel slipped out of

the motor drive shaft. The rubber on a few of the perpendicular rollers is loose and about to fall off the plastic rims. This causes the robot to shake, which can damage spinning hard disk drives or shake the sensors out of their calibrated positions.

A new set of wheels should be able to carry the weight of the robot, and enable the robot to drive over small barriers such as door sills and steel grates. This will most likely prompt a redesign of the mobile base.

7.2 Final Conclusion

A mobile robot platform has been fitted with a new software framework capable of SLAM and autonomous navigation in two dimensions. The software framework, ROS...

The SLAM system is configured to use a Kinect for Xbox 360 in combination with a LIDAR to generate 2D occupancy grid maps and 3D point cloud maps of the environments. The mapping system, RTAB-Map use visual features to The current implementation is capable of building maps over multiple sessions. Scan matching from Hector SLAM provides odometry to RTAB-Map. Using laser scans as a source of odometry was susceptible to errors in featureless areas, when the robot rounded corners and when people walked by the robot.

The navigation stack in ROS has been configured for this mobile base, and enables the robot to autonomously move towards a goal location in a 2D map.

Navigation tests on the live robot showed that the robot can navigate successfully in known and structured environments with some room manoeuvring space. Navigation performance decreased in cluttered environments, e.g. office environments with many tables placed closely together.

The Kinect and the LIDAR is currently placed in the front of the base. As the Kinect is unable to reliably measure depth any closer than $0.8m$.

Two supporting tools, an Android device and a remote OCS was also developed.

References

[ARG] ARGOS challenge. http://www.argos-challenge.com/en. Accessed: 19-05-2016.

[AS12] David A. Anisi and Charlotte Skourup. A step-wise approach to oil and gas robotics. In *Proceedings of the 2012 IFAC Workshop on Automatic Control in Offshore Oil and Gas Production*, June 2012.

[Asp13] Petter Aspunvik. Robotisert vedlikehold. Master's thesis, Dept. of Engineering Cybernetics, NTNU, 2013.

[ATE] ATEX directive: first edition of the atex 2014/34/eu guidelines. http://ec.europa.eu/growth/sectors/mechanical-engineering/atex/. Accessed: 13-05-2016.

[Bek10] Kristian Saxrud Bekken. Bevegelsesstyring av robotarm og kamera med kollisjonsunngåelse. Master's thesis, Dept. of Engineering Cybernetics, NTNU, 2010.

[Ber13] Mikael Berg. Navigation with simultaneous localization and mapping. Master's thesis, Dept. of Engineering Cybernetics, NTNU, 2013.

[BMNK13] Kai Berger, Stephan Meister, Rahul Nair, and Daniel Kondermann. *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*, chapter A State of the Art Report on Kinect Sensor Setups in Computer Vision, pages 257–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Bog16] Robert Bogue. Europe continues to lead the way in the collaborative robot business. *Industrial Robot: An International Journal*, 43(1):6–11, 2016.

[Coo11] Gerald Cook. *Mobile robots: navigation, control and remote sensing*. John Wiley & Sons, 2011.

[dep12] Sensabot: A safe and cost-effective inspection solution. *Journal of Petroleum Technology*, 64:32–34, 10 2012.

[DRC] DARPA robotics challenge. http://www.theroboticschallenge.org/overview. Accessed: 2016-04-05.

[E24] E24.no: Denne plattformen skal fjernstyres fra land. http://e24.no/energi/statoil/produksjonen-paa-valemon-er-i-gang/23366972. Accessed: 28-05-2016.

[EHS+14] Felix Endres, Jurgen Hess, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *Robotics, IEEE Transactions on*, 30(1):177–187, 2014.

[ER12a] Mohamed A. El-Reedy. Chapter 6 - corrosion protection. In Mohamed A. El-Reedy, editor, *Offshore Structures*, pages 383 – 443. Gulf Professional Publishing, Boston, 2012.

[ER12b] Mohamed A. El-Reedy. Chapter 8 - risk-based inspection technique. In Mohamed A. El-Reedy, editor, *Offshore Structures*, pages 563 – 634. Gulf Professional Publishing, Boston, 2012.

[Foo13] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[GC11] Peter Gorle and Andrew Clive. Positive impact of industrial robots on employment. Report, METRA MARTECH Limited, 2011.

[GP08] Birgit Graf and Kai Pfeiffer. Mobile robotics for offshore automation. In *Proceedings of the EURON/IARP International Workshop on Robotics for Risky Interventions and Surveillance of the Environment, Benicassim, Spain*, 2008.

[hok] URG-04LX-UG01 LIDAR specifications. http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_UG01_spec_en.pdf.

[HWB+13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at http://octomap.github.com.

[ifr] International federation of robotics - statistics. http://www.ifr.org/industrial-robots/statistics/. Accessed: 2016-04-05.

[JWA+12] J. Jamieson, L. Wilson, M. Arredondo, K. Evans, J.and Hamilton, and C Sotzing. Autonomous inspection vehicle: A new dimension in life of field operations. *Offshore Technology Conference*, April 2012.

[KFO12] Shinji Kawatsuma, Mineo Fukushima, and Takashi Okada. Emergency response by robots to fukushima daiichi accident: summary and lessons learned. *Industrial Robot: An International Journal*, 39(5):428–435, 2012.

[kina] Kinect for windows, does in work with ros? http://answers.ros.org/question/12876/kinect-for-windows/. Accessed: 10-02-2016.

[kinb] Kinect for windows, hack. http://projects.csail.mit.edu/pr2/wiki/index.php?title=Kinect_for_Windows. Accessed: 10-02-2016.

[kinc] Kinect for windows programming guide. https://msdn.microsoft.com/en-us/library/hh855348.aspx.

[Kind] Kintinous. https://github.com/mp3guy/Kintinuous. Accessed: 2016-03-21.

[KLT09] Erik Kyrkjebø, Pål Liljebäck, and Aksel A. Transeth. A robotic concept for remote inspection and maintenance on oil platforms. In *Ocean, Offshore and Arctic Engineering, ASME 2009 28th International Conference on*, 2009.

[KMP15] K. Kydd, S. Macrez, and P. Pourcel. *Autonomous Robot for Gas and Oil Sites*. Society of Petroleum Engineers, September 2015.

[KMvSK11] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[Lin15] Vegard Stjerna Lindrup. Visual sensing in mobile robots. Proj. rep., Dept. of Engineering Cybernetics, NTNU, 2015. 9th semester specialization project report.

[LM13] M. Labbe and F. Michaud. Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013.

[LM14] M. Labbe and F. Michaud. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, Sept 2014.

[MEBF+10] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *International Conference on Robotics and Automation*, 2010.

[MIM] MIMROex, mobile maintenance and inspection robot for process plants. http://www.ipa.fraunhofer.de/fileadmin/user_upload/Kompetenzen/Roboter-_und_Assistenzsysteme/Industrielle_und_gewerbliche_Servicerobotik/English_Documents/Product_sheet_MIMROex_Mobile_maintenance_and_inspection_robot_for_process_plants.pdf.

[PBB11] K. Pfeiffer, M. Bengel, and A. Bubeck. Offshore robotics - survey, implementation, outlook. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 241–246, Sept 2011.

[pre] Press release: Subsea 7 completes design and build of first commercial autonomous inspection vehicle (aiv). http://www.subsea7.com/content/dam/subsea7/Company%20News/2011/Subsea7completesdesignandbuildoffirstcommericalAIV.pdf. Accessed: 2016-04-13.

[PSM+16] Ian Peerless, Adam Serblowski, Berry Mulder, et al. A robot that removes operators from extreme environments. In *SPE International Conference and Exhibition on Health, Safety, Security, Environment, and Social Responsibility*. Society of Petroleum Engineers, 2016.

[QGS15] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS*. O'Reilly Media, Inc., December 2015.

[RCR+15] Pere Ridao, Marc Carreras, David Ribas, Pedro J. Sanz, and Gabriel Oliver. Intervention auvs: The next challenge. *Annual Reviews in Control*, 40:227 – 241, 2015.

[ROSa] ROS create package tutorial. http://wiki.ros.org/ROS/Tutorials/CreatingPackage. Accessed: 2016-02-05.

[ROSb] ROS history. http://www.ros.org/history/. Accessed: 2016-02-28.

[ROSc] ROS-industrial. http://rosindustrial.org/the-challenge/. Accessed: 2016-04-05.

[ROSd] ROS-industrial, supported hardware. http://wiki.ros.org/Industrial/supported_hardware. Accessed: 2016-04-05.

[ROSe] ROS installation. http://wiki.ros.org/indigo/Installation/Ubuntu. Accessed: 2016-02-29.

[ROSf] ROS navigation tuning guide. http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide.

[ROSg] ROS on the iss. http://www.ros.org/news/2014/09/ros-running-on-iss.html. Accessed: 2016-04-10.

[ROSh] roslaunch. http://wiki.ros.org/roslaunch. Accessed: 2016-03-15.

[rta] Setup rtab-map on your robot! http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot.

[sta] Offshore.no: Statoil vil bygge flere folkefrie plattformer. http://offshore.no/sak/63114_statoil_vil_bygge_flere_folkefrie_plattformer. Accessed: 28-05-2016.

[sub] Teknisk ukeblad: Ubemannede plattformer skal konkurrere med subsea. http://www.tu.no/artikler/ubemannede-plattformer-skal-konkurrere-med-subsea/226868. Accessed: 28-05-2016.

[SWB+14] Samuel Soldan, Jochen Welle, Thomas Barz, Andreas Kroll, and Dirk Schulz. Towards autonomous robotic systems for remote gas leak detection and localization in industrial environments. In *Field and Service Robotics*, pages 233–247. Springer Berlin Heidelberg, 2014.

[Vin14] Jan-Erik Vinnem. *Offshore Risk Assessment vol 1.: Principles, Modelling and Applications of QRA Studies*. Springer London, London, 2014.

[WA12] Jarrett Webb and James Ashley. *Beginning Kinect Programming with the Microsoft Kinect SDK*. Apress, 2012.

Appendix A

Setting Up the Project

A.1 Hardware Setup

A.1.1 Equipment List

- A computer that can be mounted on the robot. The computer must run on Linux Ubuntu (13.10 or 14.04 for ROS Indigo) and have a Bluetooth adapter.
- Two wireless routers (for example TP-Link).
- Two sinus inverters. One power inverter from Biltema and a silver colored pure sine inverter.
- One 12V Battery. The battery used in this project has a capacity of $45Ah$.
- A $230VAC/24VDC$ converter.
- One XMEGA A3BU evalation board.
- One Hokuyo URG-04LX-UG01 LIDAR.
- A Kinect for XBOX 360 or an equivalent OpenNI depth camera.

A.2 Installation

Software list

Hector SLAM for ROS Install with

```
sudo apt-get install ros-indigo-hector-slam
```

Web video server node Install with

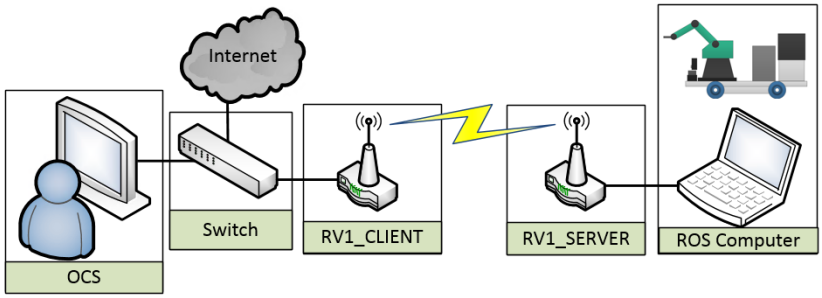


Figure A.1: Network hardware setup.

```
sudo apt-get install ros-indigo-web-video-server
```

LIDAR driver Install with

```
sudo apt-get install ros-indigo-hokuyo-node
```

Compatibility Issues

Indigo, Ubuntu etc.

A.2.1 Install Ubuntu

A.2.2 Download ROS

A.3 Configuring the Project

A.3.1 Configuring the ROS Workspace

A.3.2 Configuring the Bluetooth Connection

The Qt framework is used to simplify the implementation of the Bluetooth connection between the ROS graph and a remote device. Our ROS installation for this project already includes some variant of Qt version 4.8. While useful for creating new GUI applications, it lacks a Bluetooth API. The latest version of Qt, version 5.x, is equipped with libraries necessary for developing Bluetooth applications. This part of the guide will explain how to create a Qt 5 application which can be build by `catkin_make` and run as a *rosnode*.

1 - Install Qt5

Installing Qt5 for Linux is a straight forward procedure. Go to qt.io, and download the free version of Qt. All necessary instructions are provided. Qt5 may be installed in the home folder.

2- Enabling Qt5 in a ROS node

It is assumed that the ROS package `bluetooth_server`, is located in a catkin workspace:

```
<NAME OF CATKIN WORKSPACE>/src/bluetooth_server
```

Inside this folder, open the file ”CMakeLists.txt” and locate the following:

```
set(CMAKE_PREFIX_PATH "/home/vegard/Qt/5.5/gcc_64/lib/cmake/Qt5"
  "/home/vegard/Qt/5.5/gcc_64/lib/cmake/Qt5Core"
  "/home/vegard/Qt/5.5/gcc_64/lib/cmake/Qt5Bluetooth"
```

Change these paths to the correct paths on your system.

Appendix B
Troubleshooting

B.1 Introduction

This chapter contains proposed solutions to some of the problems that was encountered over the course of the semester. The solutions are not complete or comprehensive, but may provide some quick fixes for any students that may continue working with this project.

B.2 Hardware

B.2.1 The Wheel Fell Off!

During a test drive with the robot, the base collapsed because one of the wheel shafts had slipped out of the motor drive shaft. The solution to the problem is simply to tighten the set screw which connects the motor shaft to the wheel. The set screw is shown in figure B.1.

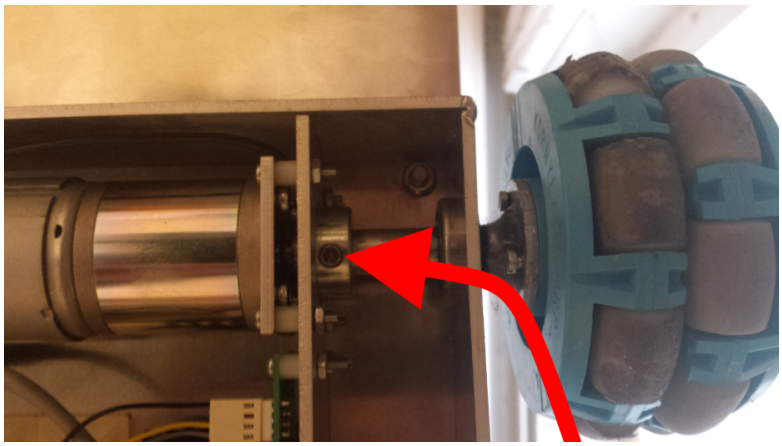


Figure B.1: The set screw which holds the wheel onto the motor drive shaft.

B.3 ROS

B.3.1 ERROR: tf2:ExtrapolationException

When running the released ROS distribution binary of `RTAB-Map` installed with `apt-get` on the robot, the node would crash after a few iterations. The error message is as follows:

```
Lookup would require extrapolation into the future, ..., when looking
up transform from frame [laser] to frame [base_link].
```

The requested transform is milliseconds ahead of "now". This issue was fixed by maintainers in March 2016[1], but was not yet integrated into the released binary. During work with this project, the problem was solved by building `RTAB-Map` from source, where the most recent fixes are included. This is a straight forward procedure, which is described on the project's GitHub repository.

B.4 Gazebo

B.4.1 Error [Node.cc.90] No namespace found

Solution: Remember to source the *gazebo* installation. In this case, with *gazebo-2.2* installed as recommended for ROS Indigo, the setup file can be sourced by typing

```
$ source /usr/share/gazebo-2.2/setup.sh
```

B.4.2 Dependency Issues When Installing *gazebo2*

This problem was encountered after removing gazebo and then typing

```
$ sudo apt-get upgrade
```

When typing

```
$ sudo apt-get install -y gazebo2
```

the installation failed because some dependencies had been upgraded to an incompatible version. To solve this, take note of the missing dependencies listed after entering

[1] https://github.com/introlab/rtabmap_ros/issues/54

the command above, open Ubuntu Software Center and select the History tab. Scroll down and locate the missing dependencies. They should have a red X next to them, indicating that they have been uninstalled. Then, enter the following command:

```
$ sudo apt-get install <NAME OF THE UNINSTALLED DEPENDENCY>
```

B.5 Ubuntu

B.5.1 Ubuntu Freezes

Sometimes during work with the project, Ubuntu would freeze and become unresponsive to keyboard input and mouse clicks. The mouse could be moved around, but was otherwise unresponsive. This event occurred exclusively when using `rviz` and displaying a camera topic as an image in the lower left corner of the GUI. The following steps from a post at askubuntu.com[2], solves the problem.

While holding Alt and SysReq (Print Screen), type R E I S U B. Press each key properly, and allow a few seconds to pass between each keystroke so that each command has time to execute. This should cause the computer to reboot, and is supposedly safer than using the power button. See the footnote for more information.

[2]What to do when Ubuntu freezes: http://askubuntu.com/questions/4408/what-should-i-do-when-ubuntu-freezes/36717#36717

Appendix C
DVD Contents

1. Blablabla